# Remix Documentation

*Release 1*

**yann300**

**Jan 25, 2022**

# New Layout Intro

Remix IDE is an open source web and desktop application. It fosters a fast development cycle and has a rich set of plugins with intuitive GUIs. Remix is used for the entire journey of contract development as well as act as a playground for learning and teaching Ethereum.

Remix IDE is part of the Remix Project which is a platform for development tools that use a plugin architecture. It encompasses sub-projects including Remix Plugin Engine, Remix Libs, and of course Remix-IDE.

Remix IDE is a powerful open source tool that helps you write Solidity contracts straight from the browser.

It is written in JavaScript and supports both usage in the browser, but run locally and in a desktop version.

Remix IDE has modules for testing, debugging and deploying of smart contracts and much more.

Remix-IDE 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. Additional tutorials are found in our Medium blog and in our tutorial tool, LearnEth located inside of Remix IDE.

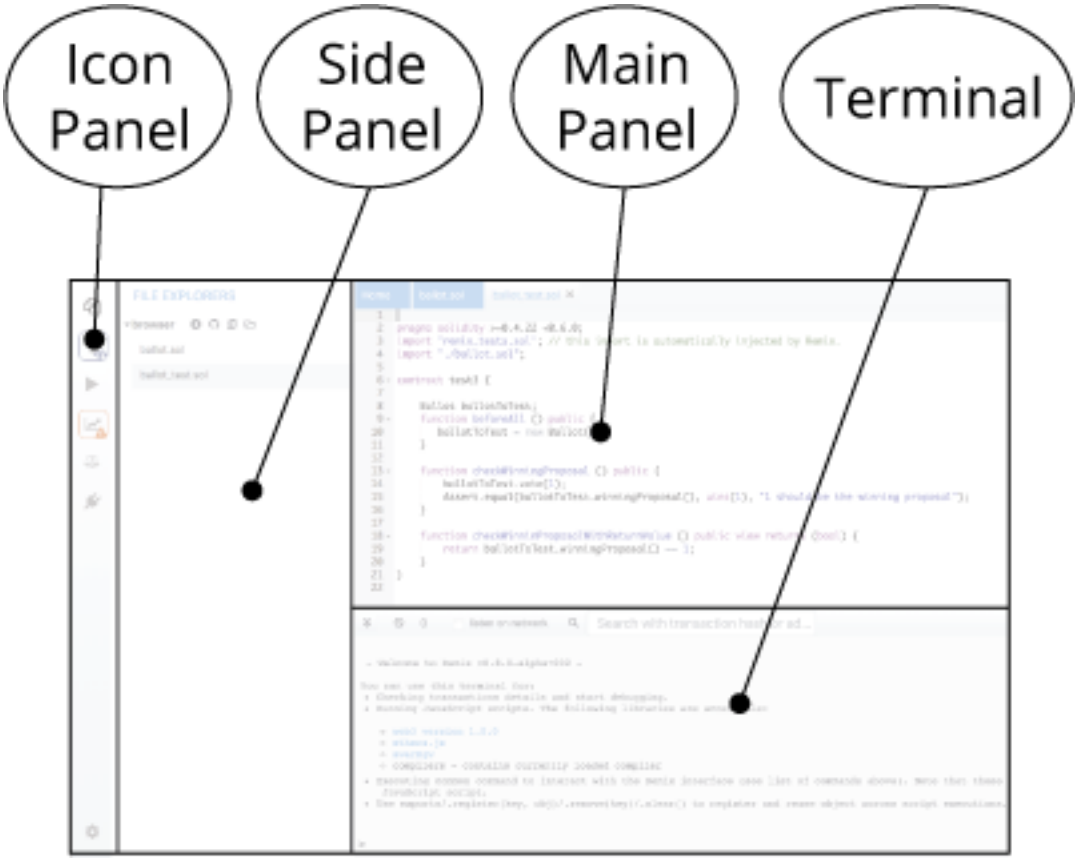
Useful links:

- Solidity documentation
- Remix alpha - The version where we test new Remix release (not stable!).
- Remix Desktop - Remix Desktop's release page.
- Remix on Github
- Remix on Medium
- Remix on Twitter
- Our Gitter support channel
- Ethereum.org's Developer resources

CHAPTER 1

# Remix-IDE Layout

## 1.1 The new structure

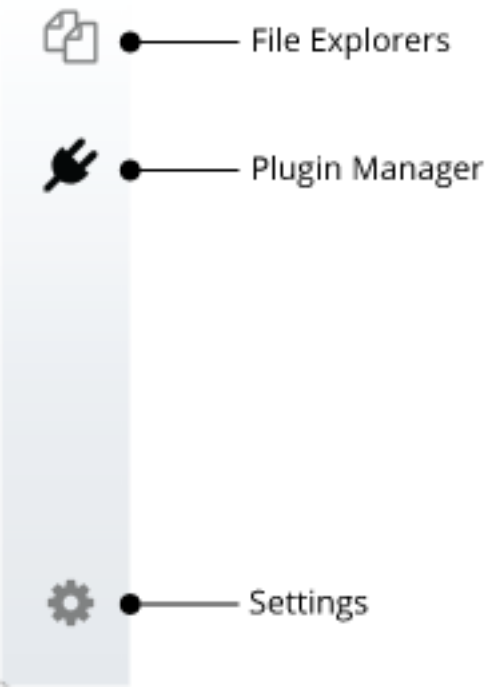



1. Icon Panel - click to change which plugin appears in the Side Panel

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

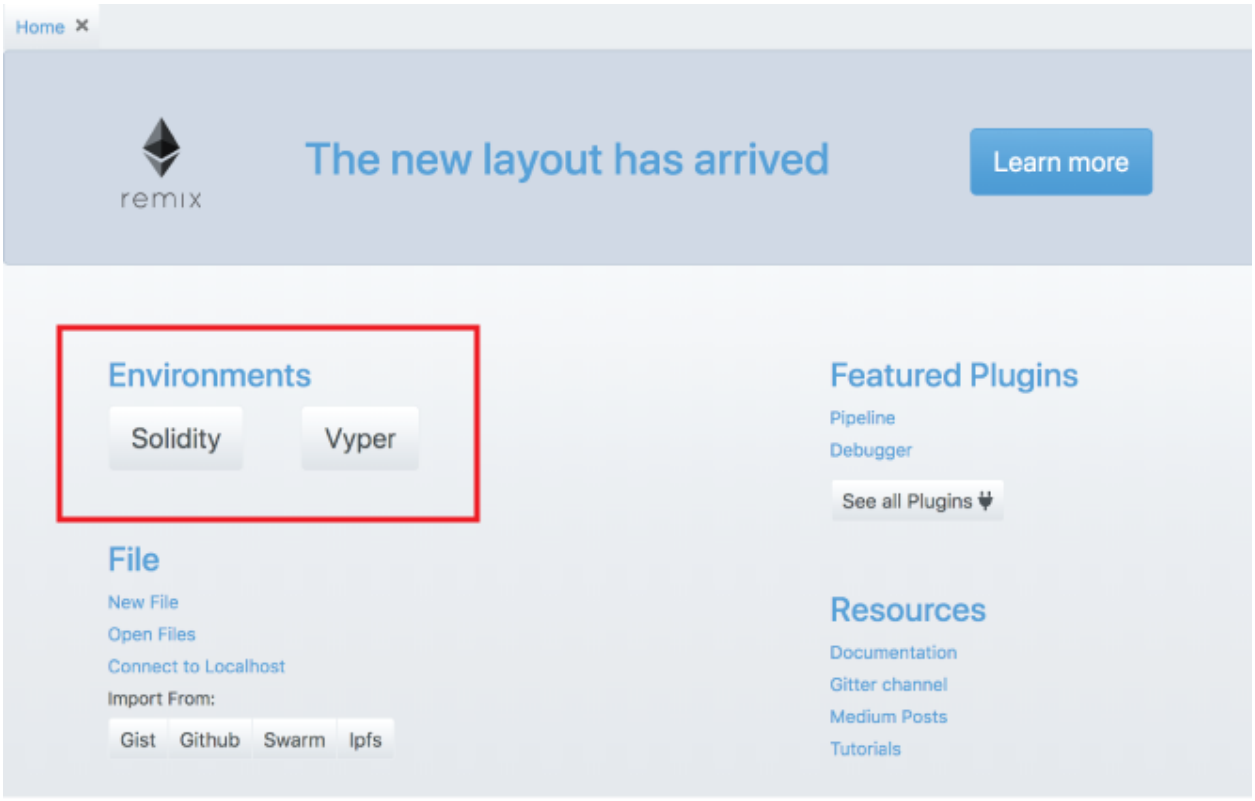
When you load remix - the icon panel show these icons by default.




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.

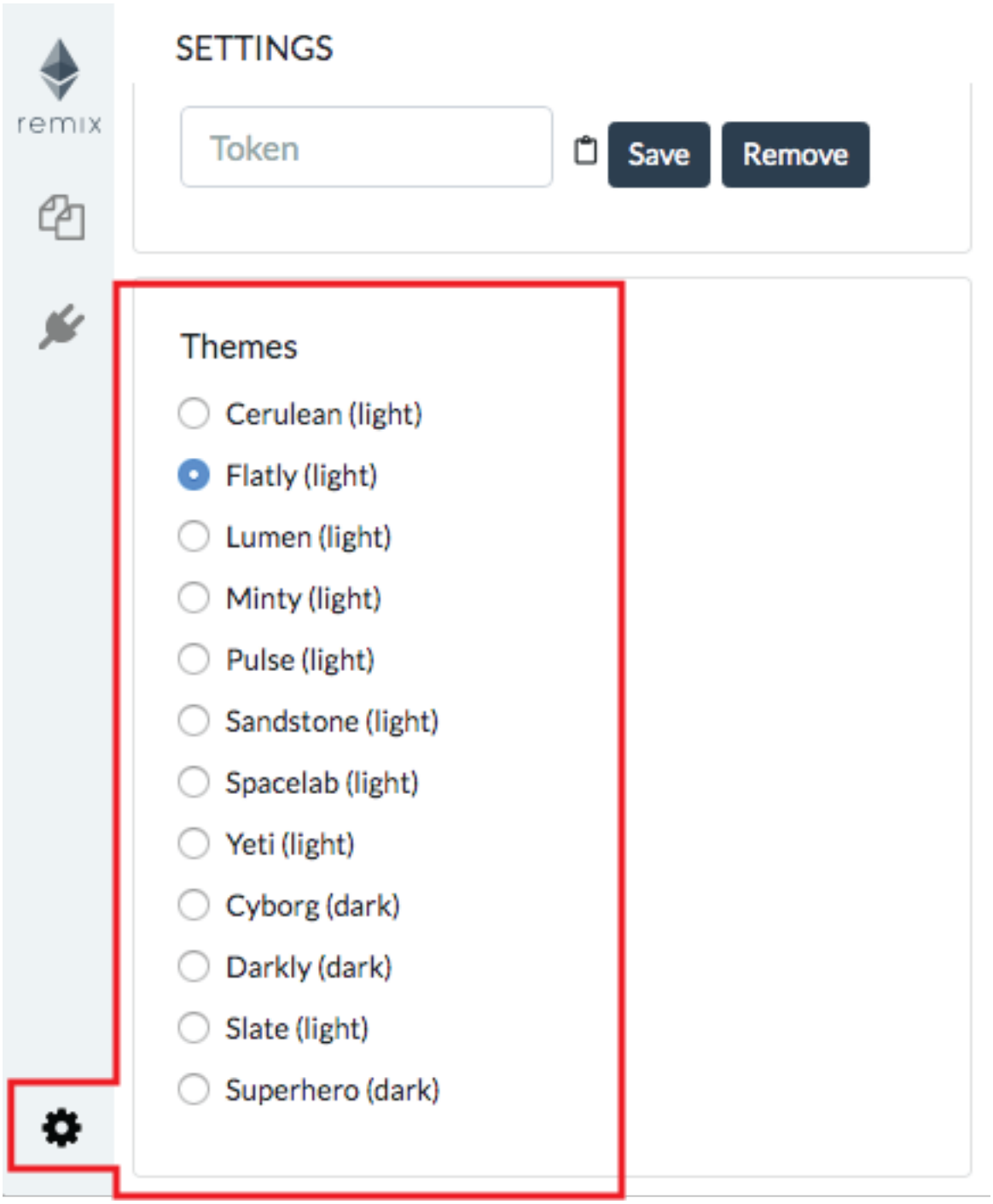
## 1.4 Plugin Manager

In order to make Remix flexible for integrating changes into its functionality and for integrating remix into other projects (yours 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 Plugin 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

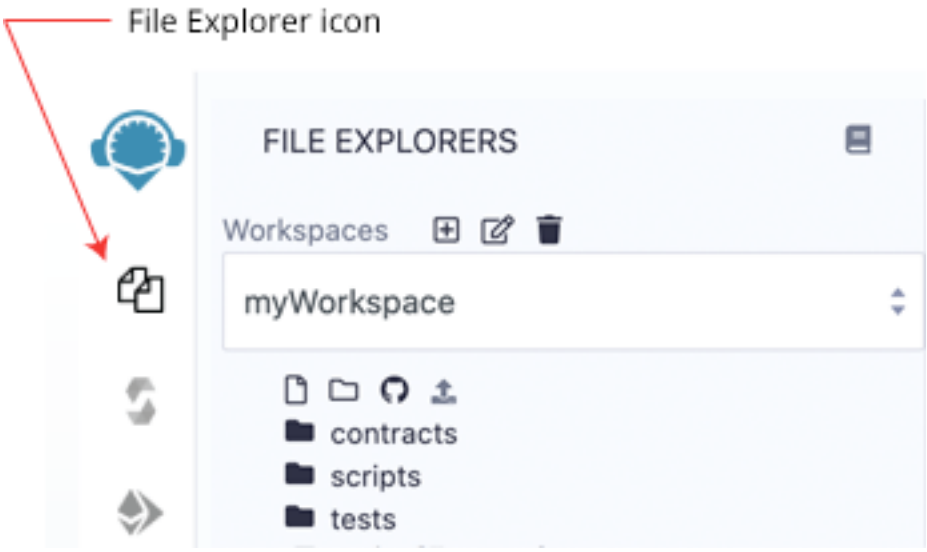
So you want to work on Remix with a dark theme or a gray theme or just a different theme that the one you are currently looking at? Go to the settings tab and at the bottom is a choice of lots of bootstrap based themes.



CHAPTER 2

# File Explorers

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




## 2.1 File Storage

By default, Remix IDE stores files in **Workspaces** which are folders in your **browser's local storage**.

**Important Note:** Clearing the browser storage will **permanently delete** all the files stored there.

If you want to use browser storage, but also to save a git repo on IPFS, use the **DGIT** plugin.

If you want to store files 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.

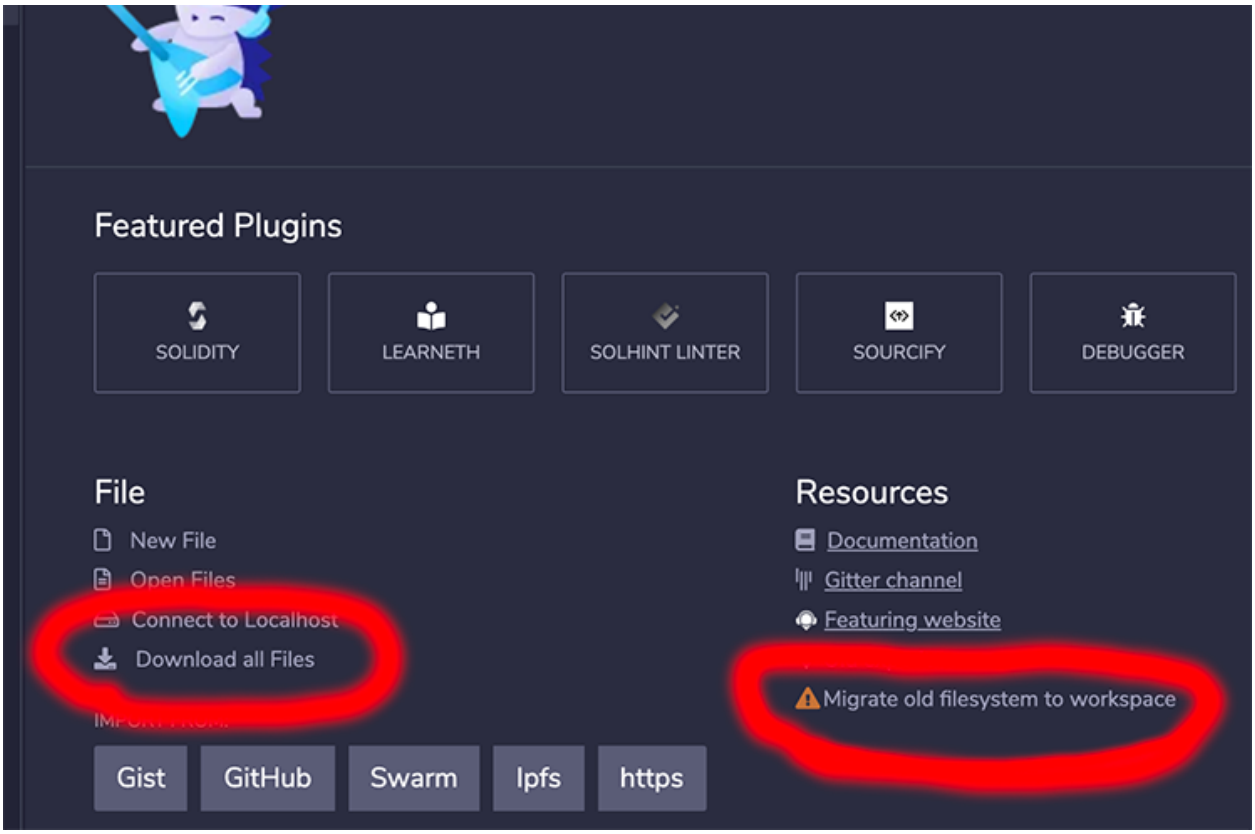
## 2.2 Workspaces

Workspaces are new as of Remix IDE v0.11.0.. Workspaces should help clean up & organize your Files Explorer by allowing you to separate your projects.

The Files Explorer's Workspaces all have a contracts folder, a scripts folder, a tests folder, and a README.txt.

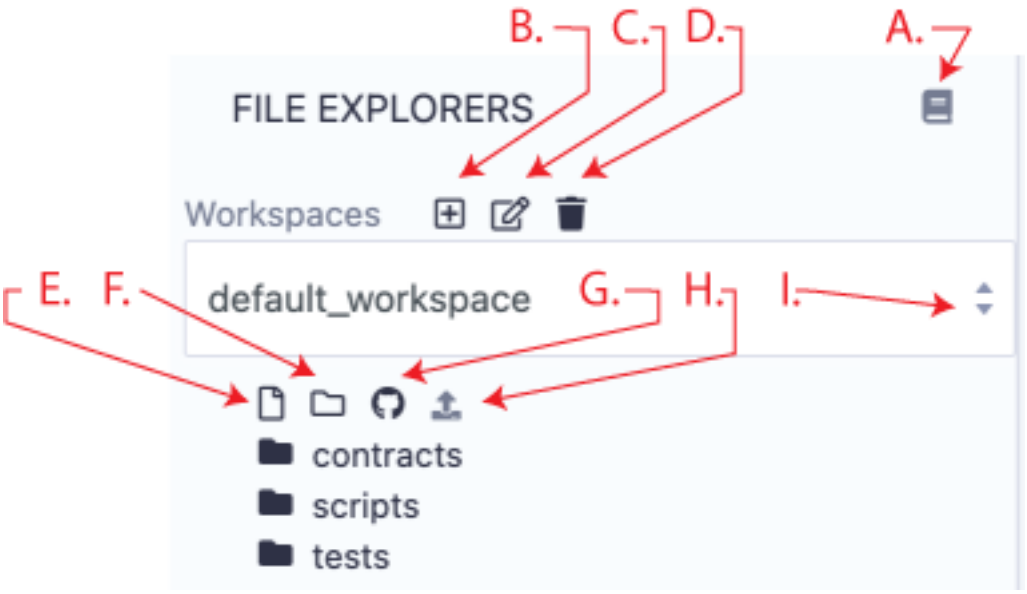
### 2.2.1 Migrating your files to Workspaces

If you had files stored in browser storage in versions of Remix before v0.11.0 to get these files, you will need to click on the **Migrate old filesystem to workspace** link the Home Tab. After you click on that, you'll get a modal asking if you want to migrate and download or just migrate the files.




Also note the **Download all files** link on the left of the Home tab. This link will download all files in your browser's local storage to your downloads folder.

## 2.3 Files Explorer Tour




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

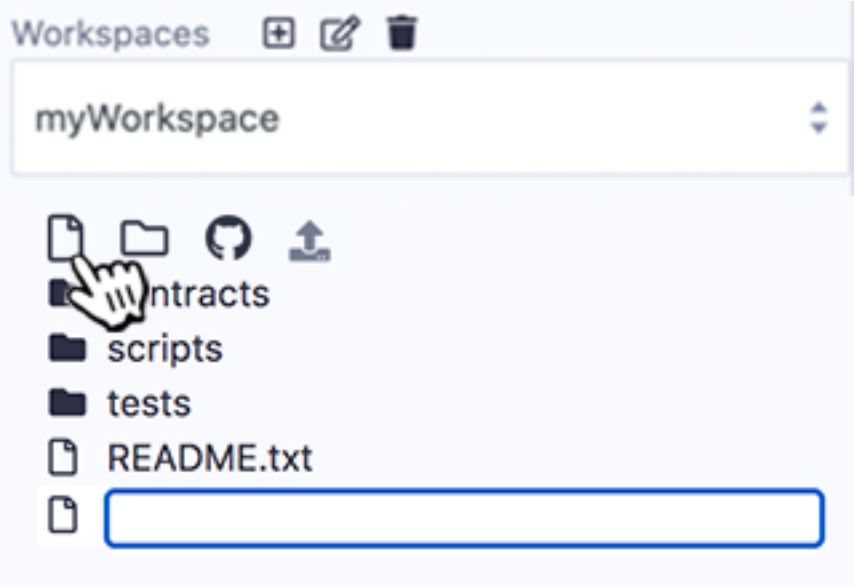
The book icon - **A.** is the link to the documentation (this page).

### 2.3.1 Workspace Manipulation

Add a Workspace **B.** Rename a Workspace **C.** Delete a Workspace **D.**

## 2.4 File Manipulation

Click on the new file icon (**E.**), an input for a new the file's name will appear in the **Explorer**. Once a name is entered, the new empty file will open in the Editor.




When you click on the new file icon, the new file will be placed in the currently selected folder. If a file and not a folder is selected then the new file will be placed in that file's folder. And if nothing is selected, then the file will be placed in the root of the current workspace's folder. Or to be brief — just be mindful of what folder it lands in.

## 2.5 Create a folder

The icon (marked **F.** above) creates a new folder in the current workspace.

## 2.6 Publish to Gist

The icon (marked **G.** above) publishes all files from the current Workspace to a gist. Only file in the root of **browser** will be published. Files in subfolders will not be published to the Gist.

Gist API **requires** users to be authenticated to be able to publish a gist.

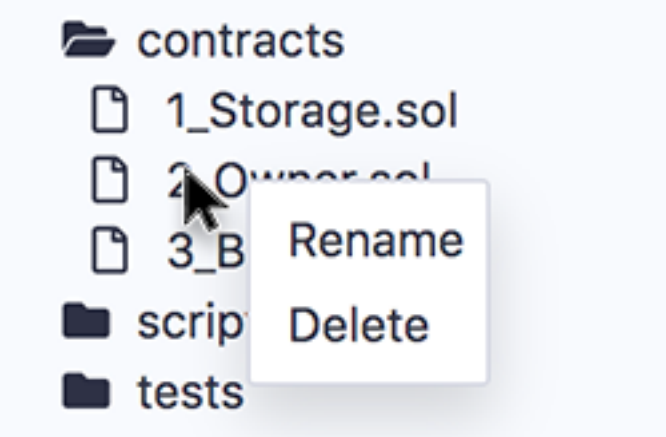
Click this link to Github tokens setup and select Generate new token. Then check the **Create gists** checkbox and generate a new token. Also make sure you check the box to enable the creation of Gists with this token.

Take the token and paste it in Remix's **Settings** module in the **Github Access Token** section. And then click Save. **For the moment**, after saving, in order for the token to be registered, you will need to refresh Remix. In an upcoming release, it will not be necessary to do the refresh.

## 2.7 Upload to Browser Storage

Click the icon marked **H.** to upload a file from your computer's file system to your browser's local storage.

## 2.8 Right-Click on a File

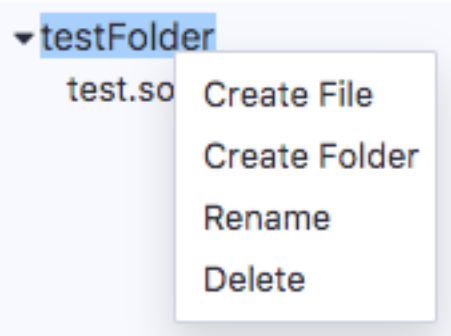



Right-clicking on a file will give you a context menu — offering you the possibility to delete or rename the file.

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

## 2.9 Right-Click on a Folder

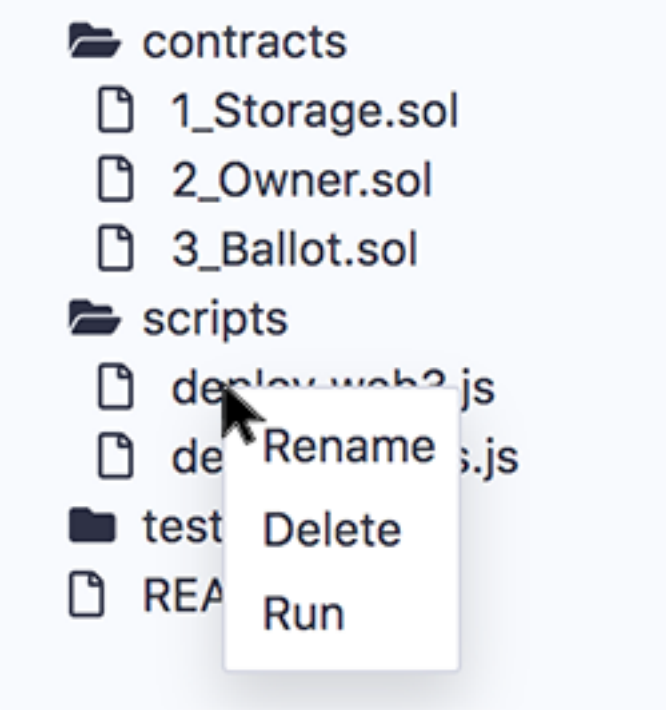
To create a file with the context menu, right-click on a folder to get the **Create File** option. A file will be created inside that folder.




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.

## 2.10 Right-Click on a Script

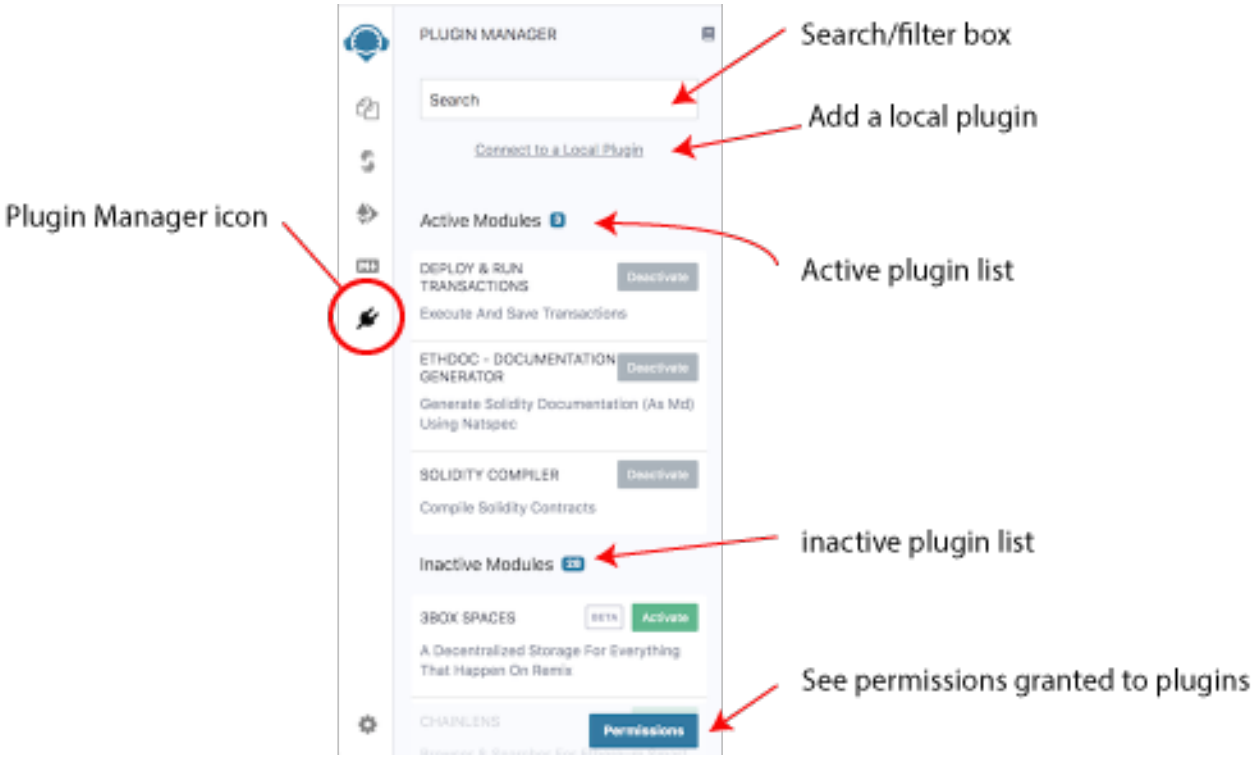



Right-click on any file with a .js extension to get the **Run** option in the context menu to run the script. The **Run** in the context menu is a shortcut. The other way to get a script to run is to:

1. Click on the script to make it the active tab in the editor
2. Input the command `remix.exeCurrent()` in the console.

CHAPTER 3

# Plugin Manager

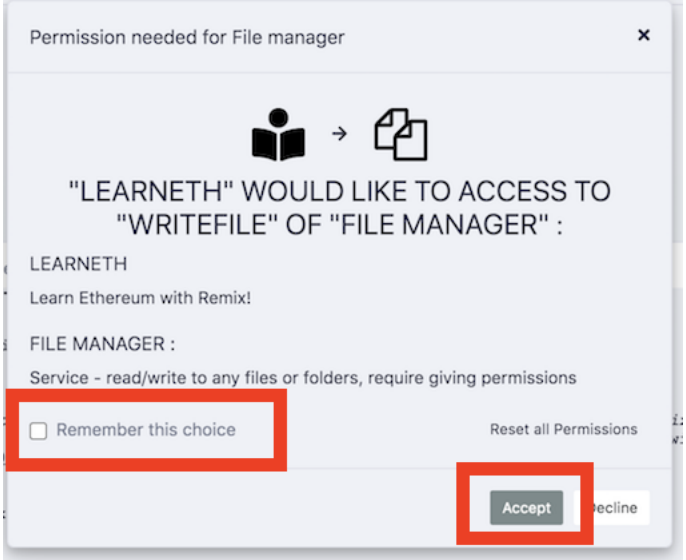



In Remix IDE you only load the functionality you need. Controlling which plugins are active or inactive happens in the **Plugin Manager**.

This plugin architecture has made it possible to integrate tools made by the Remix team with tools made by external teams. This architecture also allows Remix or just parts of Remix to be integrated into other projects.

## 3.1 Manage permissions

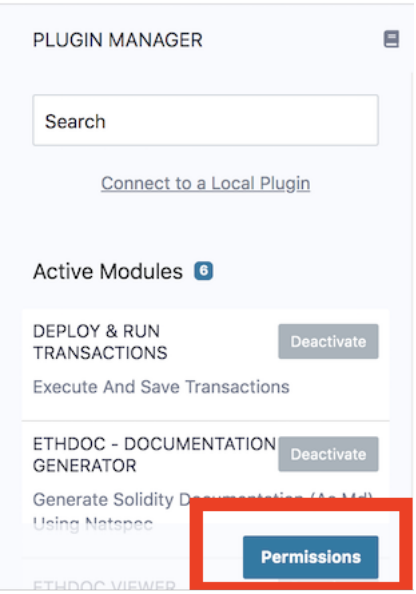
When plugins need to access other plugins for their operation, a modal will appear to ask you for permission.



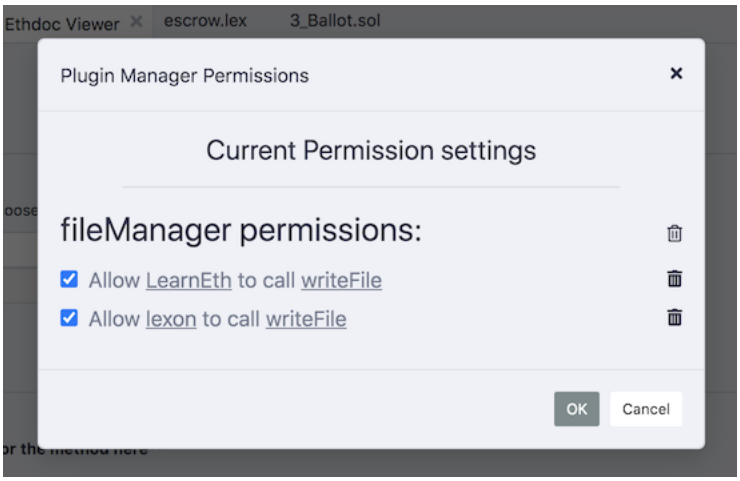

Often, the same plugin will want to do the same action multiple times. So when granting permission, its helpful to click the **Remember this choice** box. If you don't, you might get this modal repeatedly popping up.

## 3.2 View permissions

You can view the permissions that you have granted to plugins by clicking on the **Permissions** button at the bottom of the **Plugin Manager**.

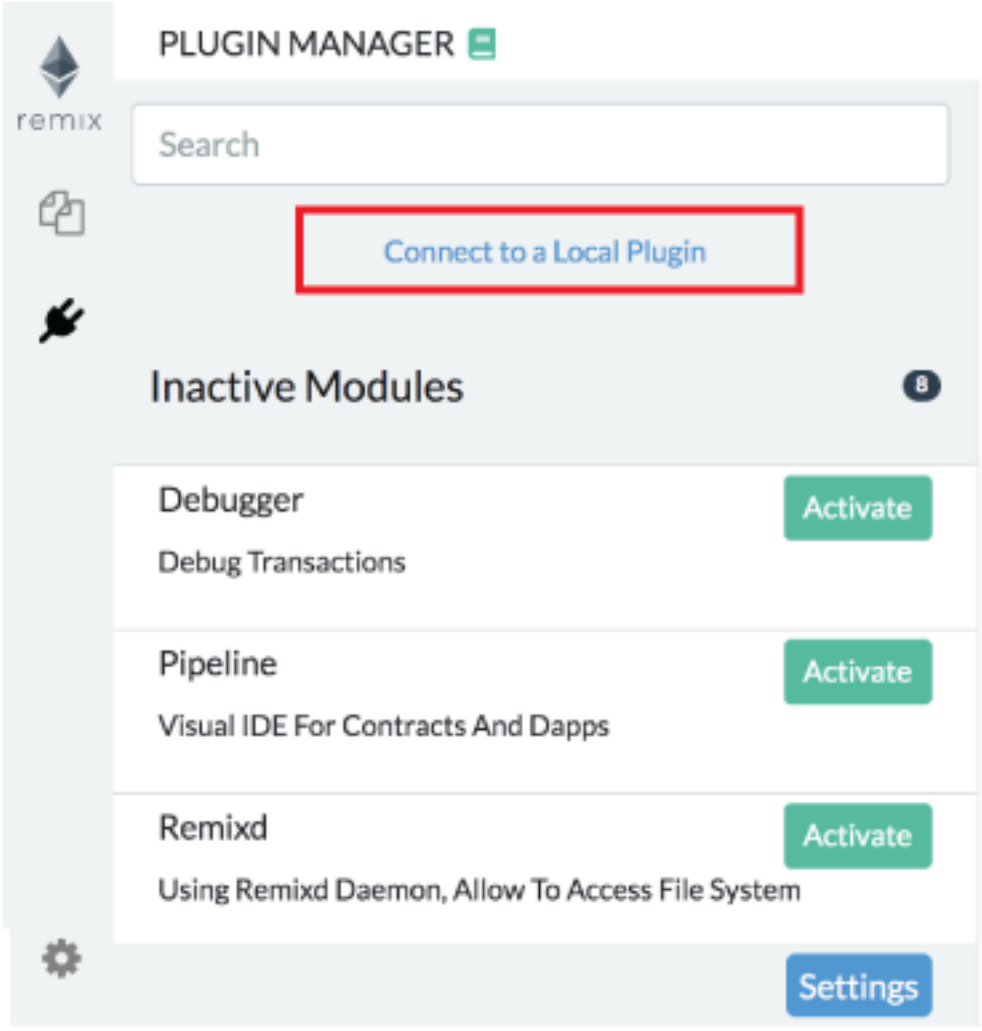



A modal will appear like the one below where you can view and erase the granted permission.




## 3.3 Plugin Devs: Load a local plugin

A plugin in development can be loaded into Remix IDE by clicking the “Connect to a Local Plugin” link at the top of the Plugin Manager panel.



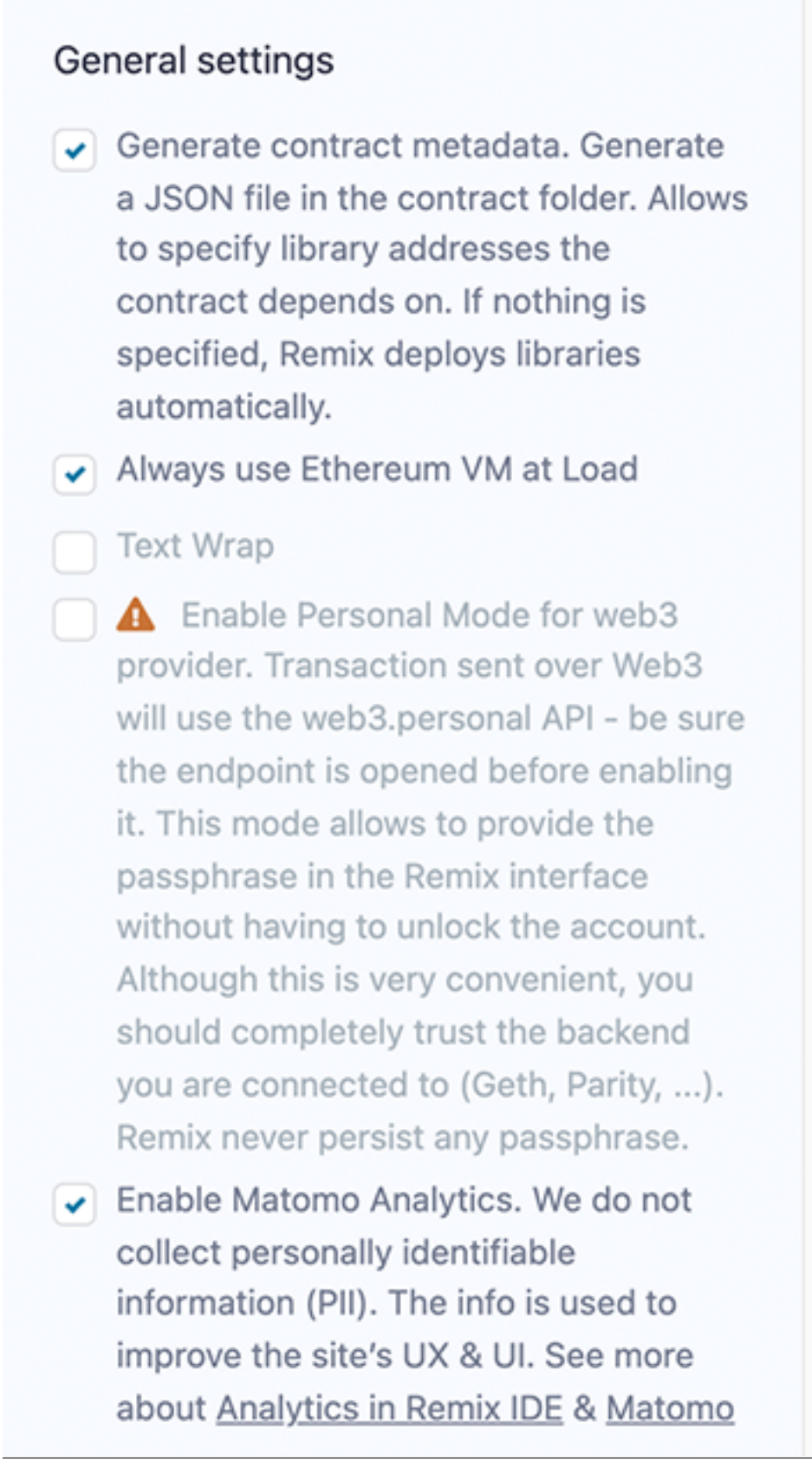

To learn more about how to create your own plugin, go to the README of remix-plugin repo.

CHAPTER 4

# Settings

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

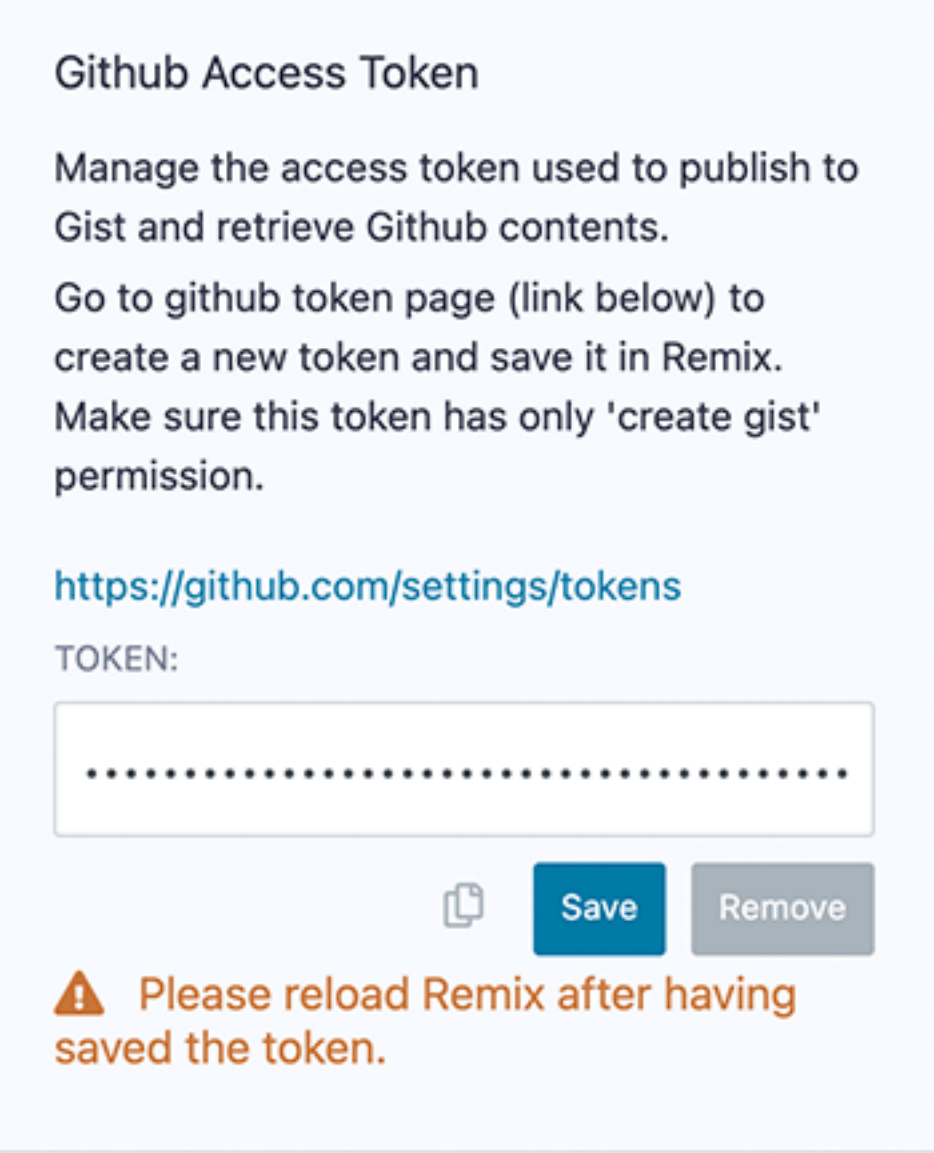
## 4.1 General Settings



- Generate contract metadata is used for deploying with libraries. See our blog post on the subject: Deploying with Libraries
- Always use Ethereum VM at Load: will make the Javascript VM the selected **environment** when Remix loads.
- Text wrap: controls if the text in the editor should be wrapped.
- Personal mode: can be used when one is connecting to a **local node**. It is used to have Remix temporarily save the passphrase - so that you don't need to **unlock** the account in GETH. Remix will not persist the passphrase - so if you refresh your browser the passphrase will be gone.
- Matomo Analytics: This is where you can turn off and on your approval for us to use Matomo. We do not collect any personally identifiable information (PII) and our reports are public. See our blog post on the subject.

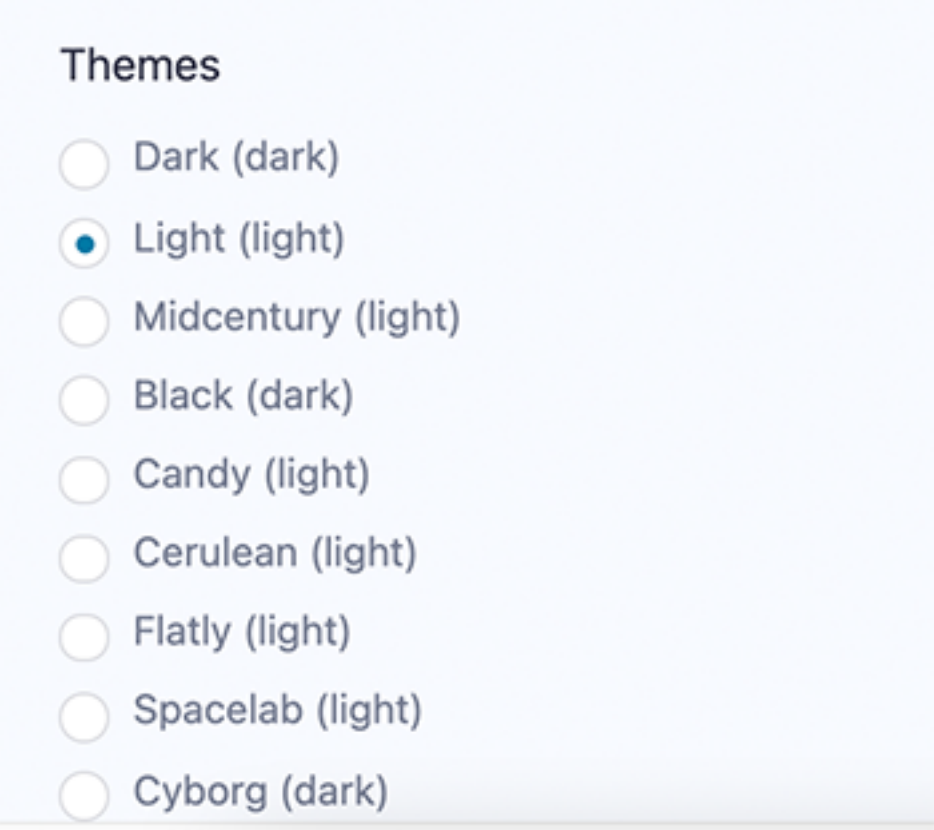
## 4.2 Github Access Token

For creating GISTs as well as using other Github functions, it is necessary to put in an access token. https://github.com/settings/tokens




## 4.3 Themes

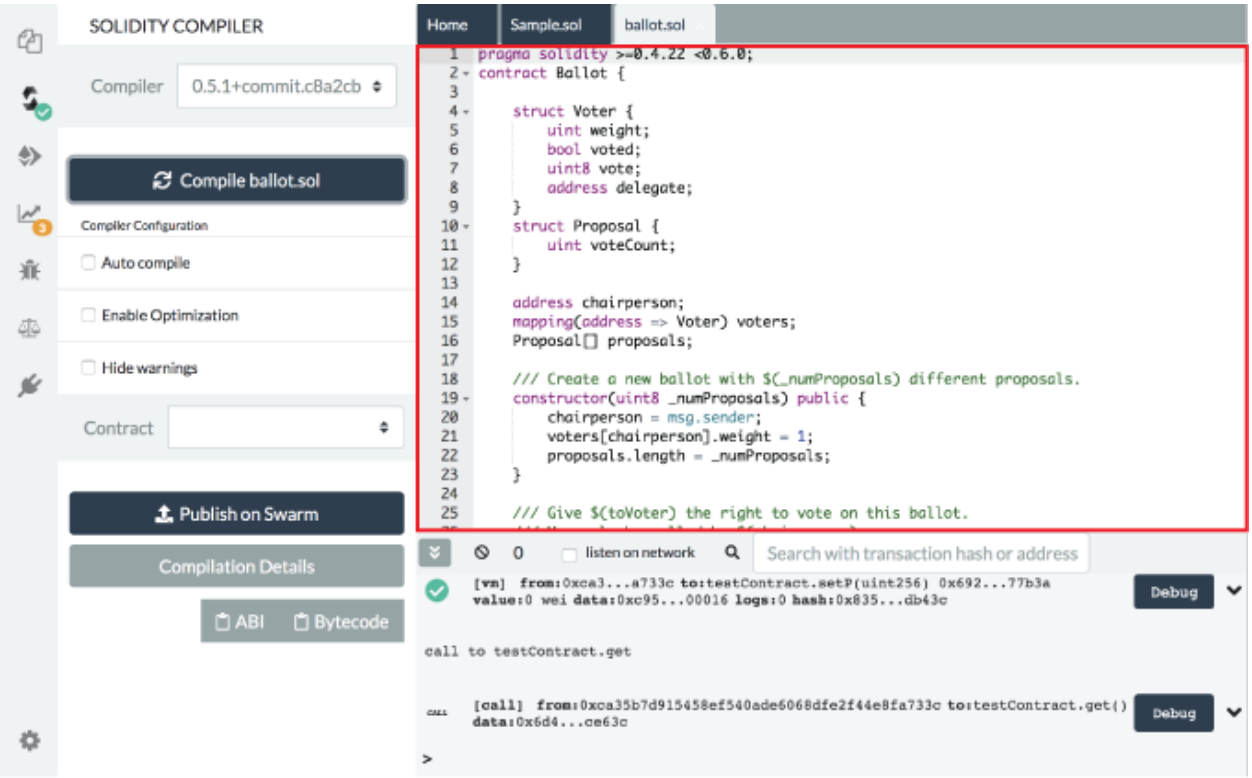
Choose themes here. The Dark & Light themes are the themes that the Remix team actively maintains.


Themes
Dark (dark)
Light (light)
Midcentury (light)
Black (dark)
Candy (light)
Cerulean (light)
Flatly (light)
Spacelab (light)
Cyborg (dark)

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.



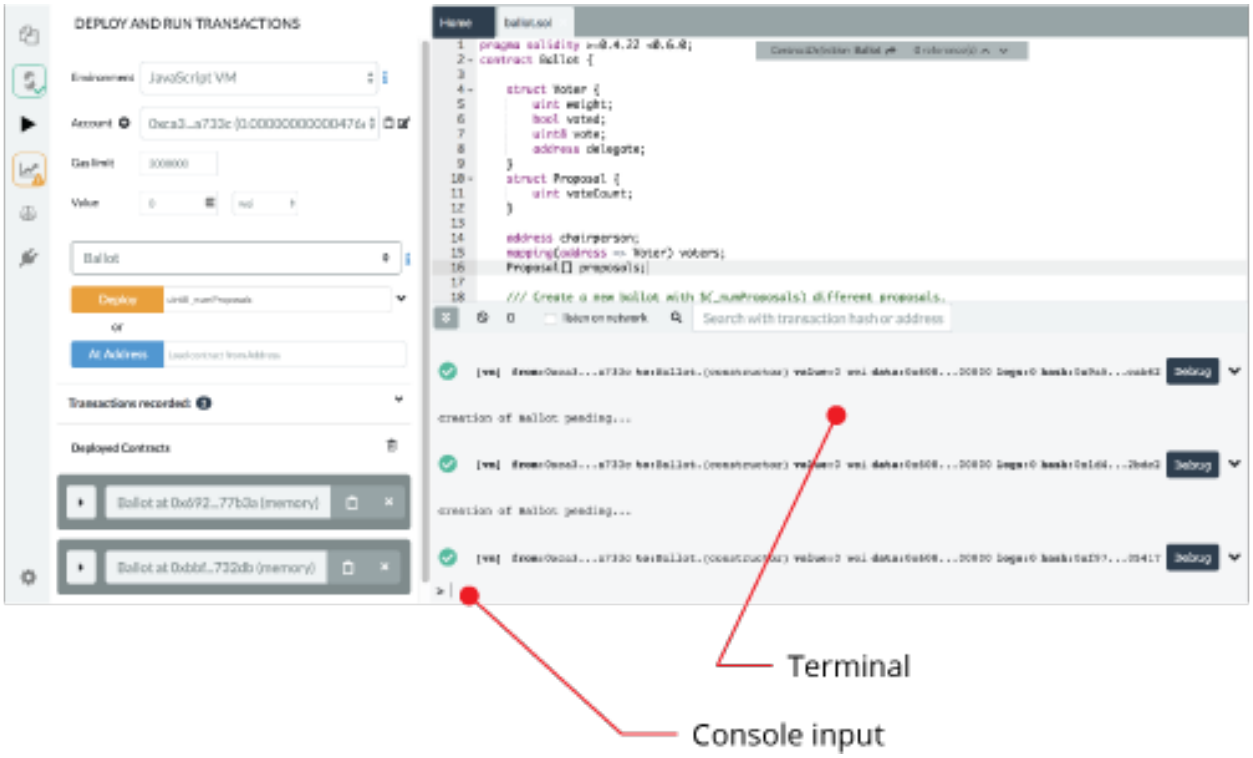

Here's the list of some important features:

- 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 Solidity Compiler.

Compiling is triggered when you click the compile button ( **F. 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 ( **D. in image below**).

## 7.1 Solidity versions & Remix functionality

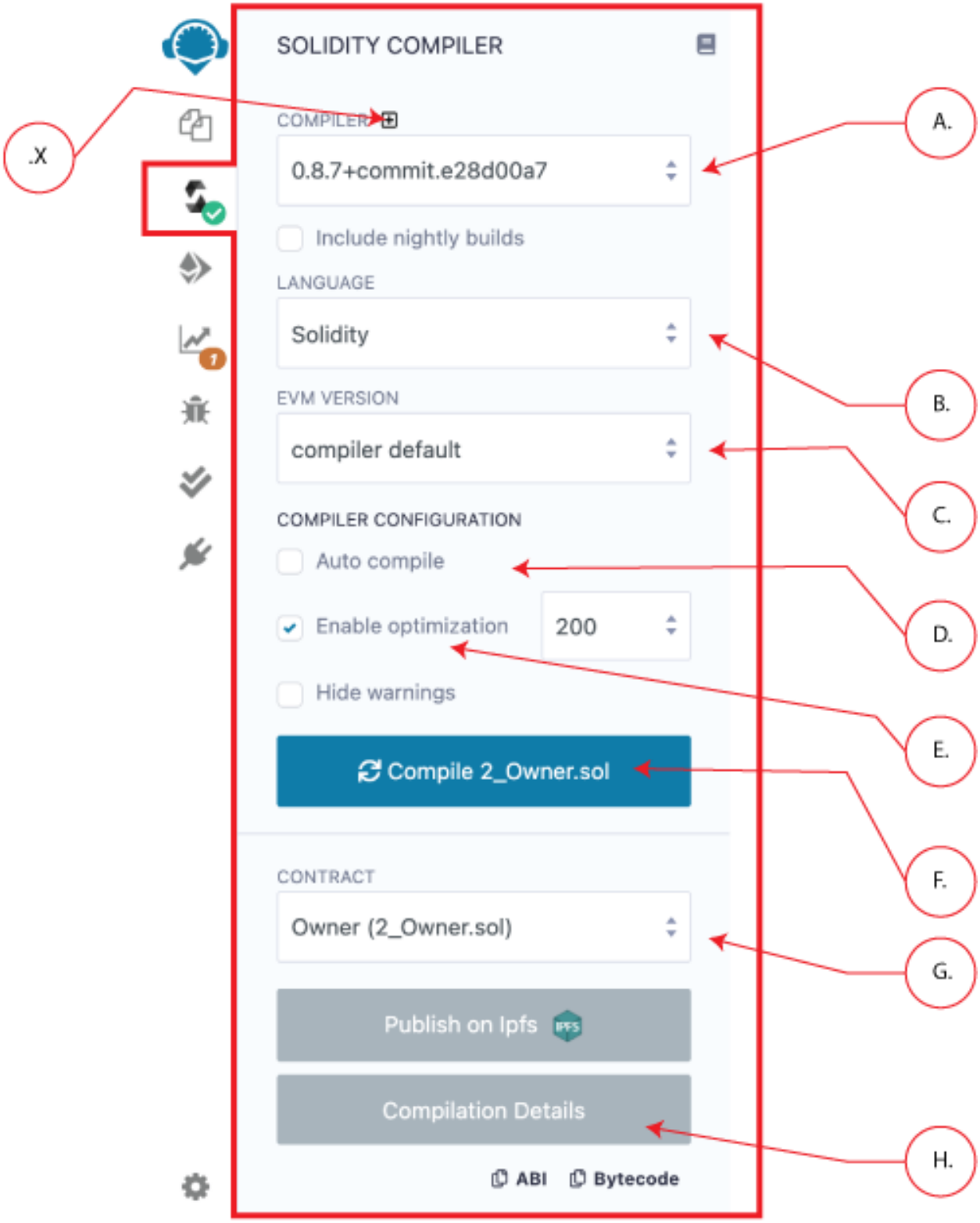
Since the Solidity version `0.5.7`, it is possible to compile `Yul` files. Please read the (solidity documentation about Yul) which contain some code examples. You can use the language dropdown ( **B. in image below**) to switch the language. **This dropdown list is only available for versions greater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

You can compile (and deploy) contracts with versions of Solidity **older than 0.4.12**. However, the older compilers use a legacy AST — which we no longer support. Consequently, some plugins may not work and some functionality - like source highlighting in the Editor may only be partially working.

## 7.2 Select an Ethereum fork

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.

To see the name of the hard fork used in the current compilation, click the "Compilation Details" button( **H. in image below**) and in the `Metadata` section will be a sub-section called **settings**. Open up the **settings** to see the hard fork's name.


SOLIDITY COMPILER
COMPILER
0.8.7+commit.e28d00a7
Include nightly builds
LANGUAGE
Solidity
EVM VERSION
compiler default
COMPILER CONFIGURATION
Auto compile
Enable optimization
200
Hide warnings
Compile 2_Owner.sol
CONTRACT
Owner (2_Owner.sol)
Publish on Ipfs
Compilation Details
ABI
Bytecode
.X
A.
B.
C.
D.
E.
F.
G.
H.

## 7.3 Auto Compile

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

## 7.4 Enable optimization

According to the the Solidity Docs, "the optimizer tries to simplify complicated expressions, which reduces both code size and execution cost, i.e., it can reduce gas needed for contract deployment as well as for external calls made to the contract."

For recent versions of Solidity, it is recommended to enable optimization .

To learn more about optimization, ( **F. in the image**) visit the Solidity docs on the optimizer.

To the right of the **Enable optimization** checkbox is the box to input the number of Optimization runs. The default value is 200.

You may ask — "What is the right number of runs for my contract?" And the Solidity docs say:

> If you want the initial contract deployment to be cheaper and the later function executions to be more expensive, set it to –optimize-runs=1. If you expect many transactions and do not care for higher deployment cost and output size, set –optimize-runs to a high number.

To learn more about the optimization runs, visit the Solidity docs about Optimizer options.

## 7.5 Compilation Details and Publishing

Because a solidity file can include multiple contracts and because contracts can import other contracts, **multiple contracts are often compiled**. **However**, only 1 contract's compilation details can be retrieved at a time.

To select the desired contract, use the **Contract select box** ( **G. in the image**).

Using the publish button, you can upload your contract to Swarm (only non abstract contracts can be published) & IPFS.

When publishing contracts that import other contract, the main contract and all of its imported contracts will be published - each to their own address.

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

After a contract is published, **a modal will pop up**. This modal contains the contract's address as well as the addreses of the contracts that it imported and the address of the contract's **metadata**.

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

## 7.6 Compilation Errors and Warning

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)

## 7.7 Custom Solidity Compilers

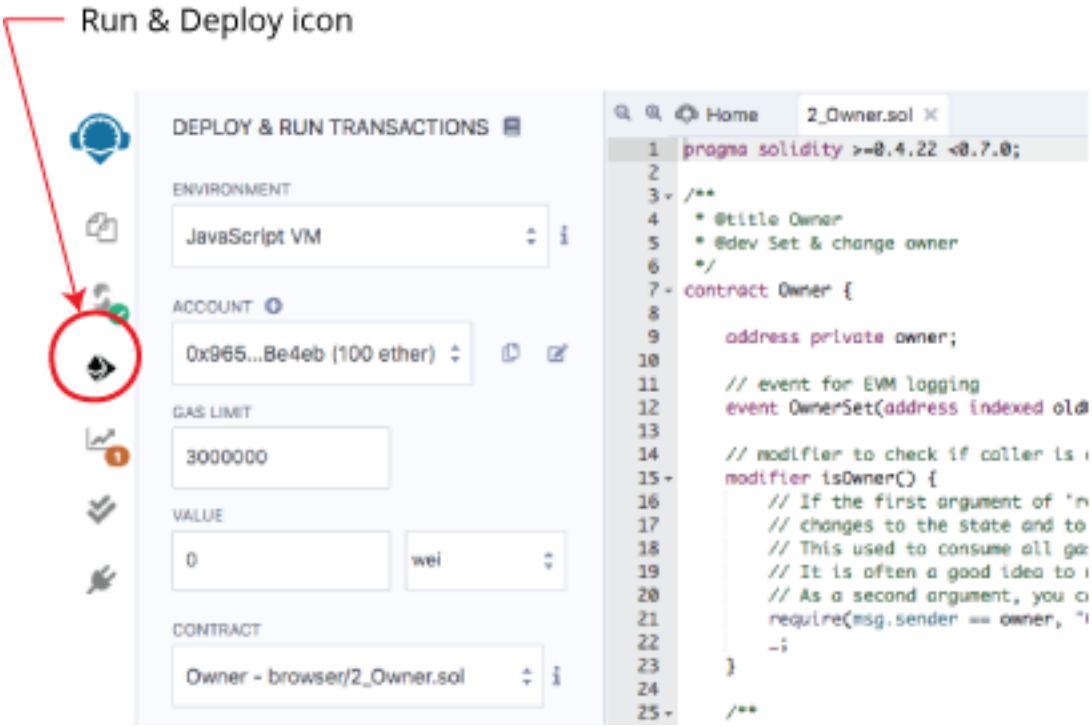
For those writing your own custom solidity compiler, you can import that by clicking the **+** button (**X. in the image**) to open a modal where you can input the url of the compiler to be loaded.

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 –http –http.corsdomain https://remix.ethereum.org**

Also see Geth Docs about the http server

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

**geth –http –http.corsdomain=”https://remix.ethereum.org” –http.api 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 –http.corsdomain with the url of Remix that you are using.

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

**geth –http –http.corsdomain=”package://a7df6d3c223593f3550b35e90d7b0b1f.mod” –http.api 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 –http.corsdomain with a wildcard: `--http.corsdomain *`

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. `--http.corsdomain 'https://remix-alpha.ethereum.org'`

Only use `--http.corsdomain *` 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.

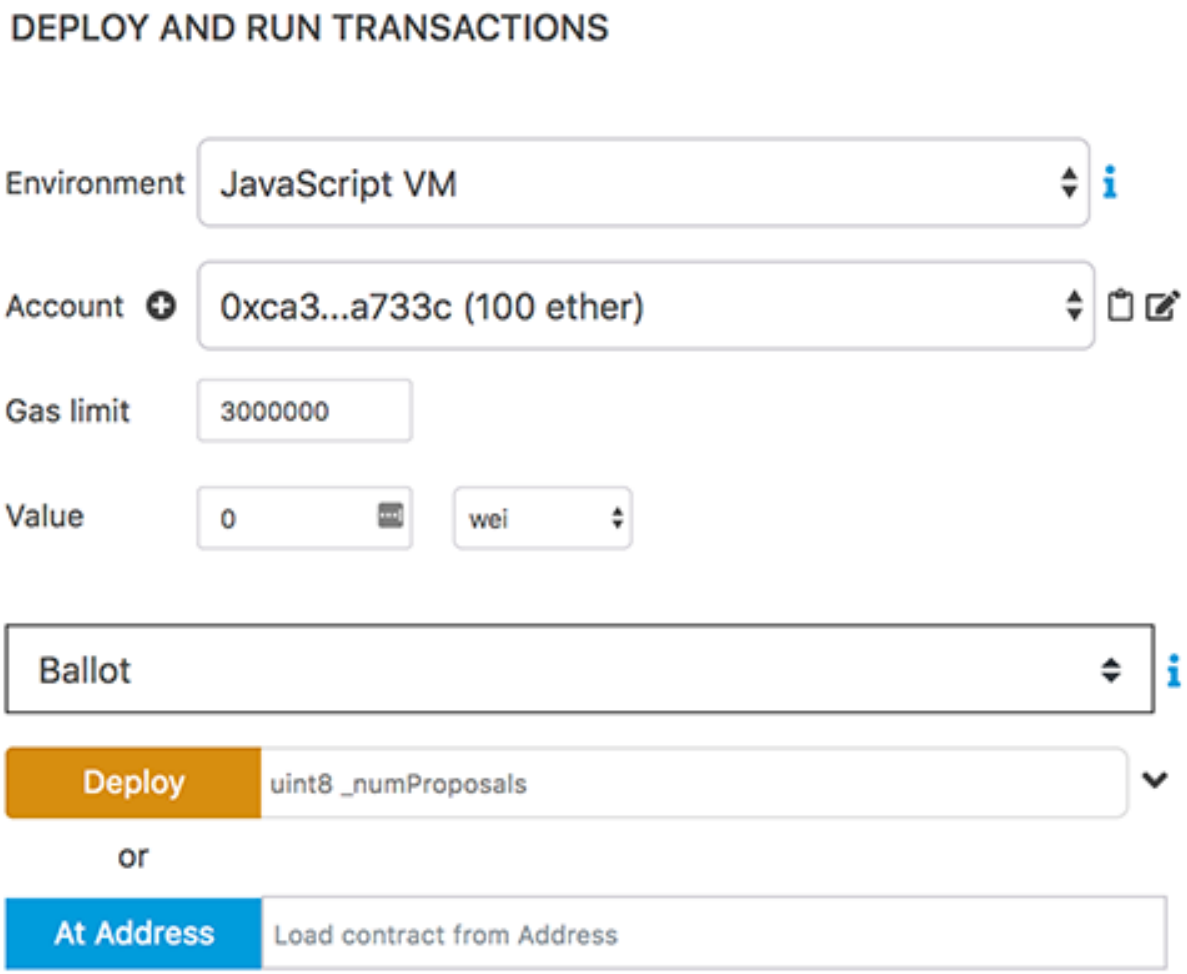
## 8.4 Gas Limit:

- 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** field is always reset to 0 after each transaction execution. The **Value** field is **NOT** for gas.




## 8.6 Deploy & AtAddress

- In the image above, the select box is set to **Ballot**. This select box will contain the list of compiled contracts.
- `Deploy` sends 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:** If the contract's constructor function has parameters, you need to specify them.

- `At Address` is used to access a contract that has already been deployed. Because the contract is already deployed, accessing a contract with **AtAddress** does not cost gas.

**Note:** When using AtAddress, be sure you trust the contract at that address.

To use **AtAddress**, you need to have the **source code** or **ABI** of the deployed contract **in the active tab** of the editor. When using the source code, it must be compiled with the same compilation settings as the deployed contract that you are trying access.

## 8.7 Using the ABI with AtAddress

The **ABI** is a JSON array which describes the contract's 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.

Make sure this file is the active tab in the editor. Then, in the field next to `At Address`, input the contract's address and click on `At Address`. If successful, an instance of the contract will appear below - in the list of **Deployed Contracts**.

**Note:** To generate the ABI, in the Solidity compiler, after a contract is compiled, click on the **Compilation Details** button. A modal will come up with that contains the ABI among other info.

## 8.8 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).

## 8.9 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.

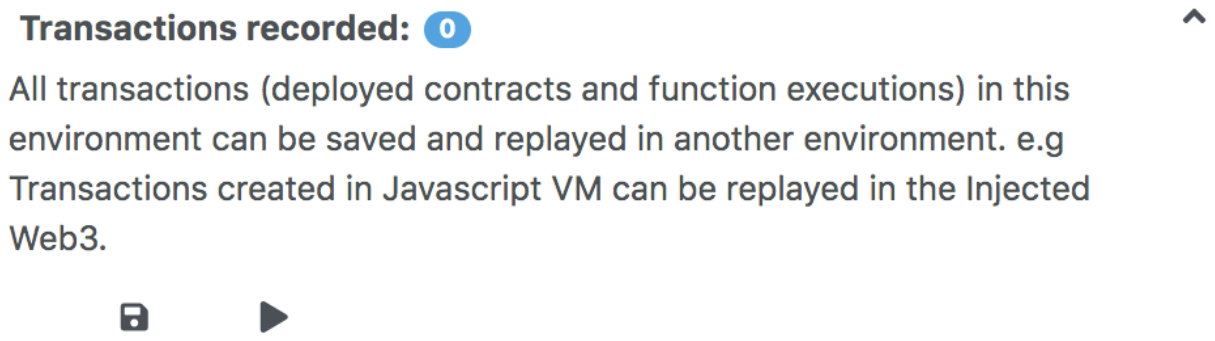
For instance:

- 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":
↪"6060604052341560e57600080fd5b60968061001c6000396000f300606060405260043610603f576000357c0100000000
↪",
        "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":
↪"606060405260405160208061102b1833981016040528080519060200190919050508060008190555505061027a806100376
↪_browser/ballot.sol:testLib____________
↪636d4ce63c60006040516020015260405181630fffffff167c0100000000000000000000000000000000000000000000000000
↪",
        "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

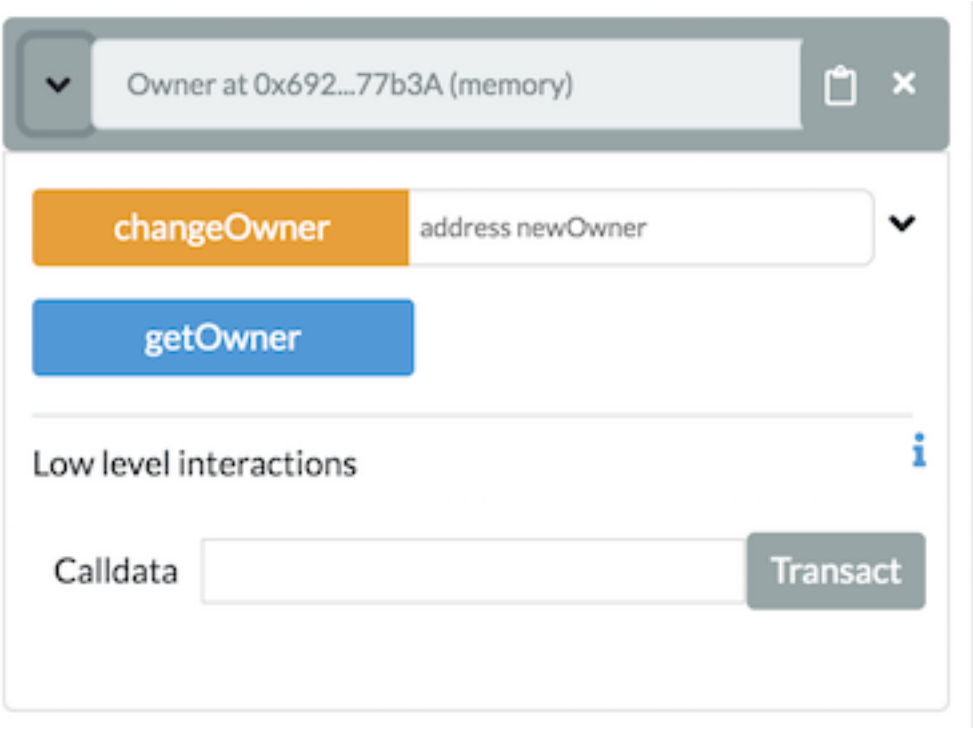
# Run & Deploy (part 2)

## 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).

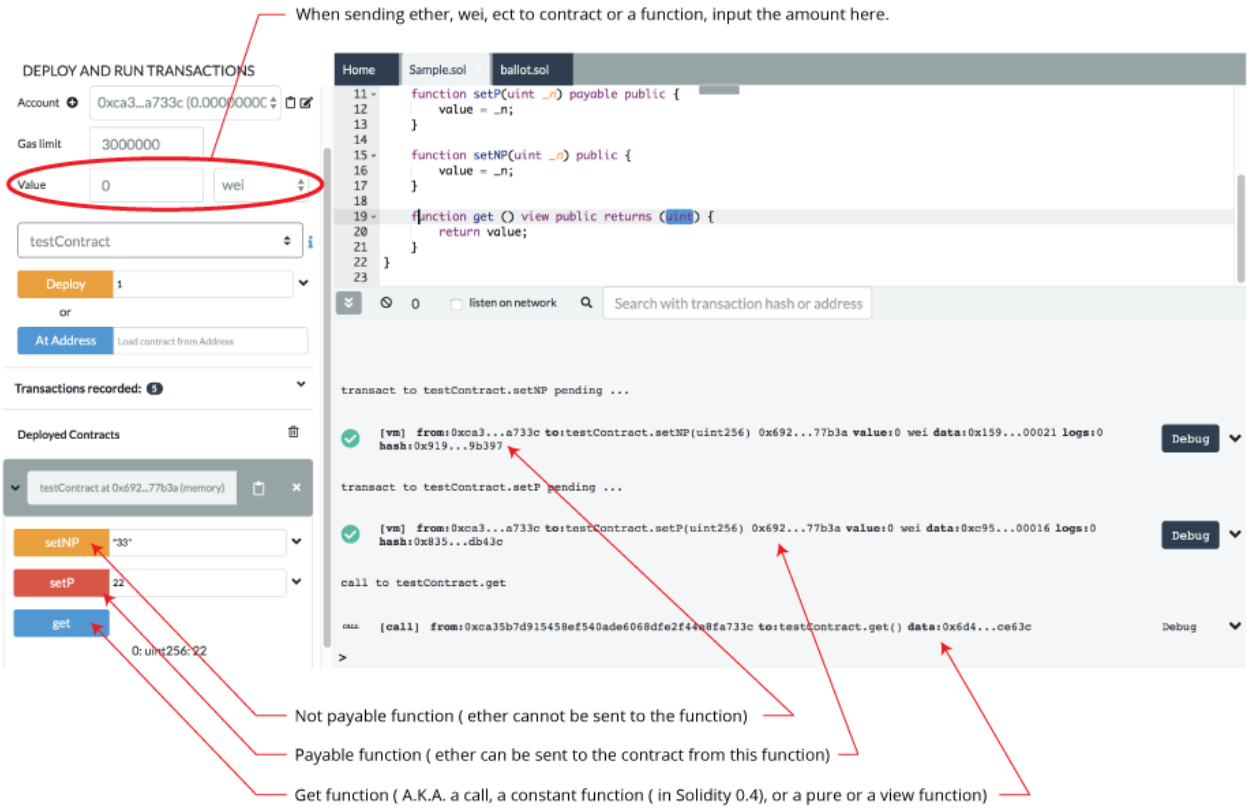
The deployed contract appears but is in its collapsed form.


Click the sideways caret to open it up.




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

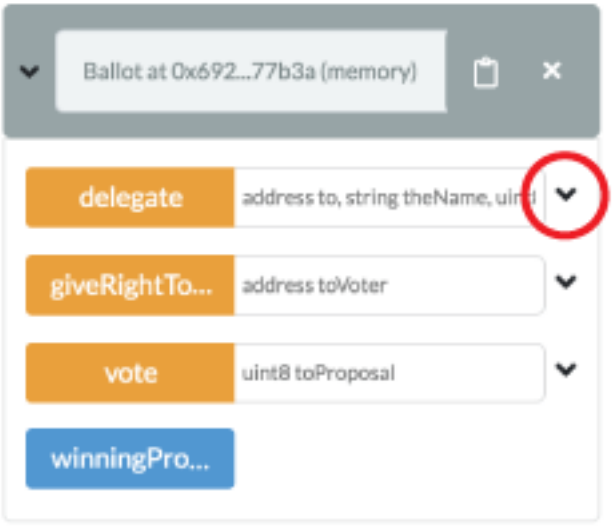
- Functions that are `constant` or `pure` functions in Solidity have a blue buttons. Clicking one of this type does not create a new transaction. So clicking will not cause state changes - it will only return a value stored in the contract - so it won't cost you anything in gas fees.
- Functions that 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

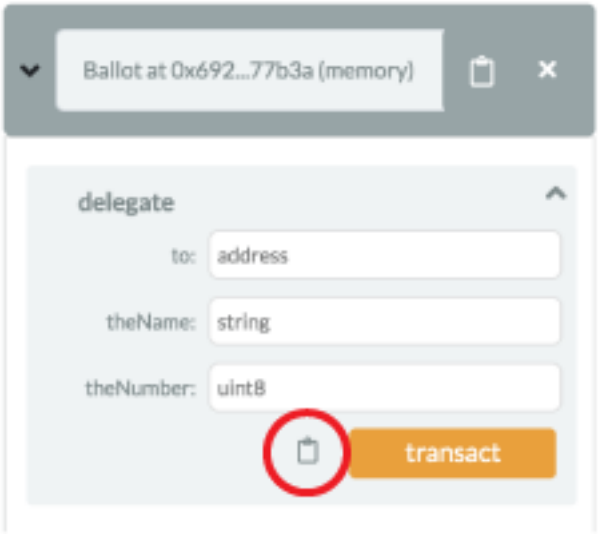
(Inputting all the parameters in a single input box)

- 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.

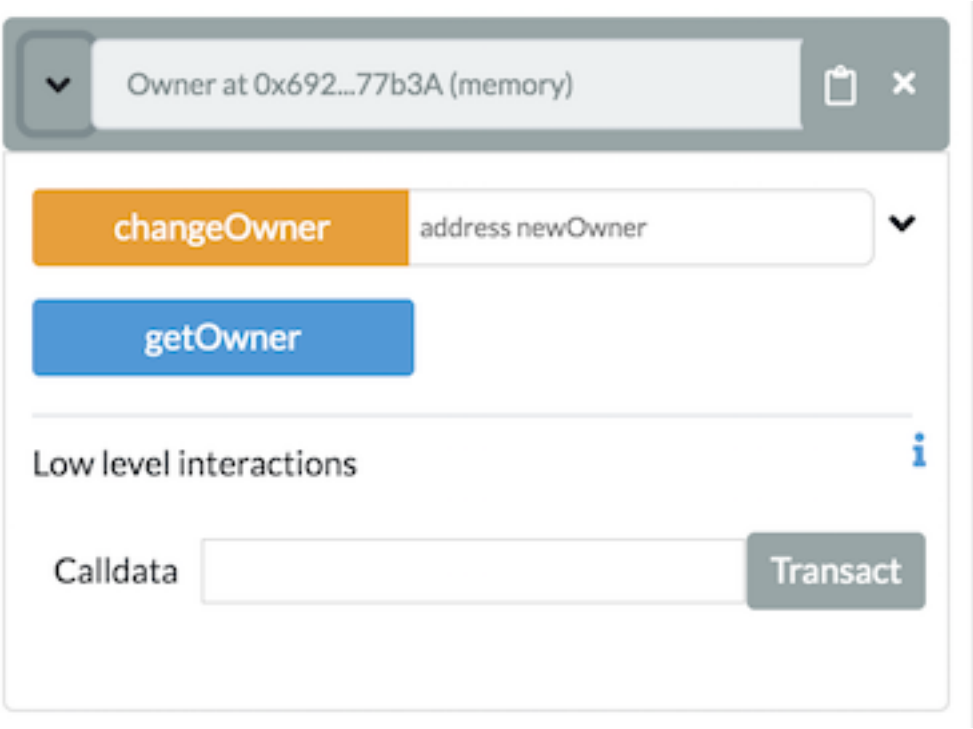
Clicking the clipboard icon will encode the inputs and will copy them. Only a valid set of inputs can be encoded.

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

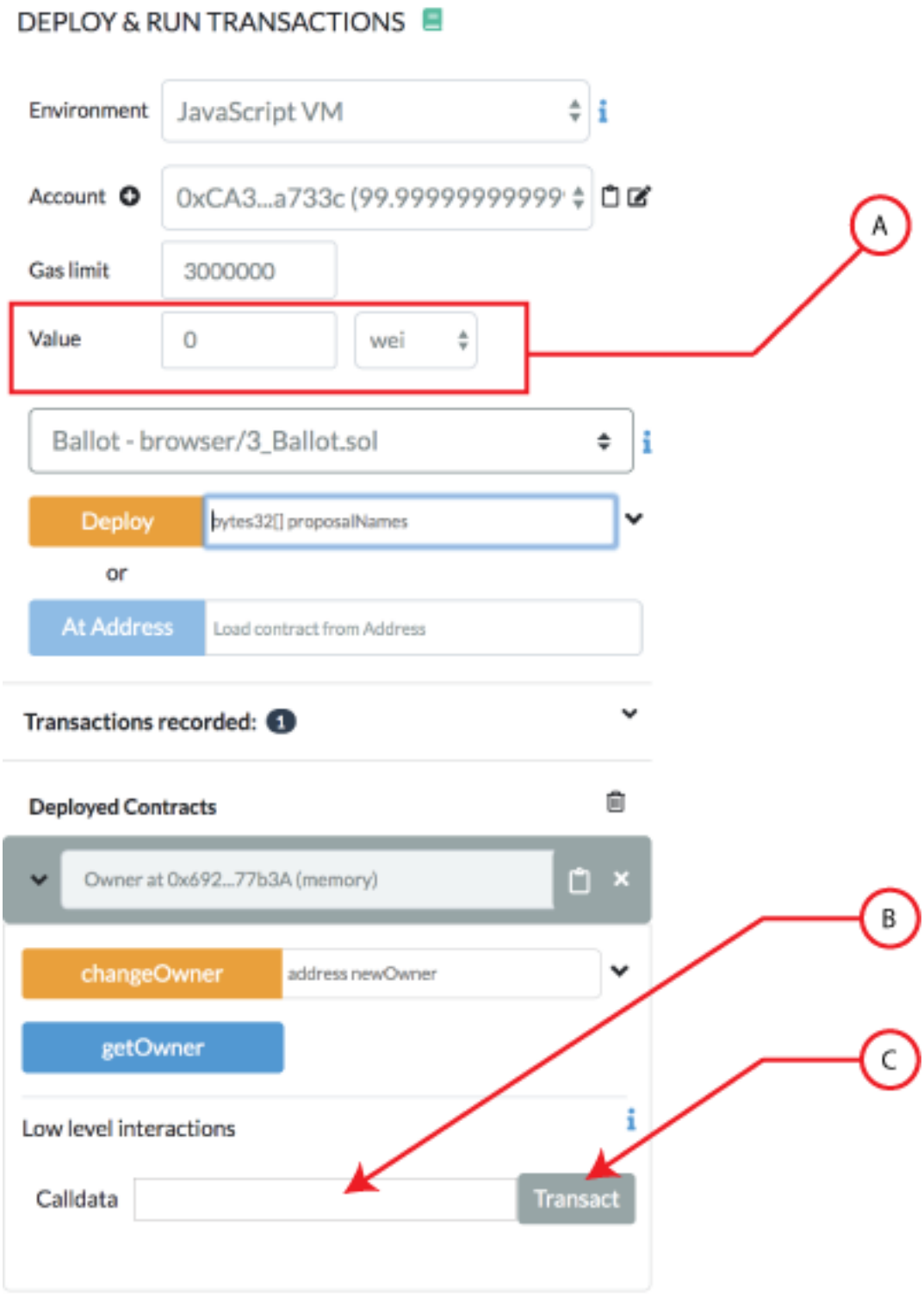
Low level interactions are used to send funds or calldata or funds & calldata to a contract through the **receive()** or **fallback()** function. Typically, you should only need to implement the fallback function if you are following an upgrade or proxy pattern.

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

The Debugger shows the contract's state while stepping through a transaction.

It can be used on transactions created on Remix or by providing a transaction's address. The latter assumes that you have the contract's source code or that you have input the address of a verified contract.

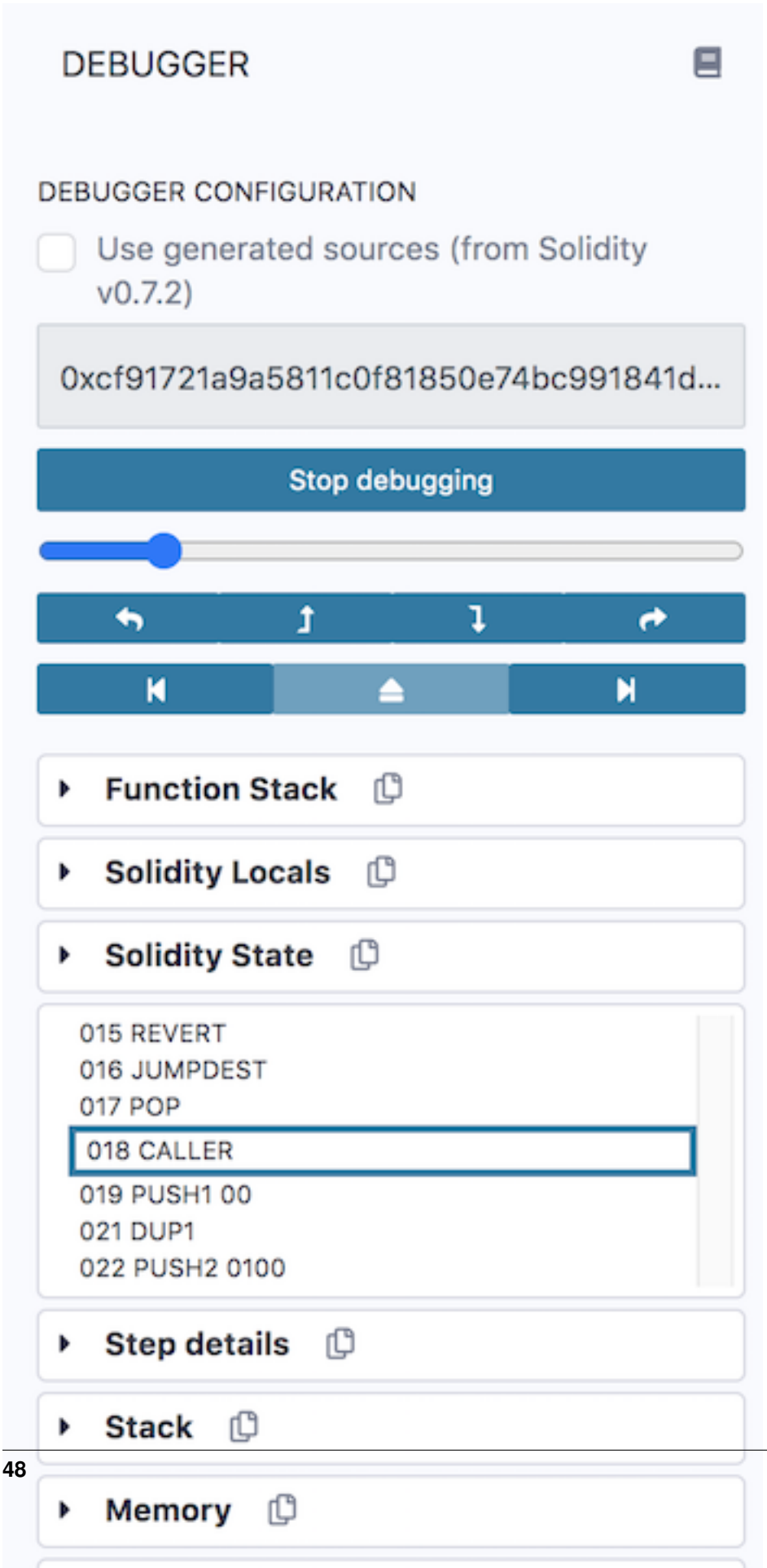
To start a debugging session either:

- **Click** the debug button in the Terminal when a successful or failed transaction appears there. The Debugger will be activated and will gain the focus in the **Side Panel**.
- **Activate** the Debugger in the Plugin Manager and then click the bug in the icon panel. To start the debugging session, input the address of a deployed transaction - while having the source code in the editor and then click the **Start debugging** button.

The debugger will highlight the relevant code in the Editor. If you want to go back to editing the code without the Debugger's highlights, then click the **Stop Debugging** button.

To learn more about how to use this tool go to the Debugging Transactions page.

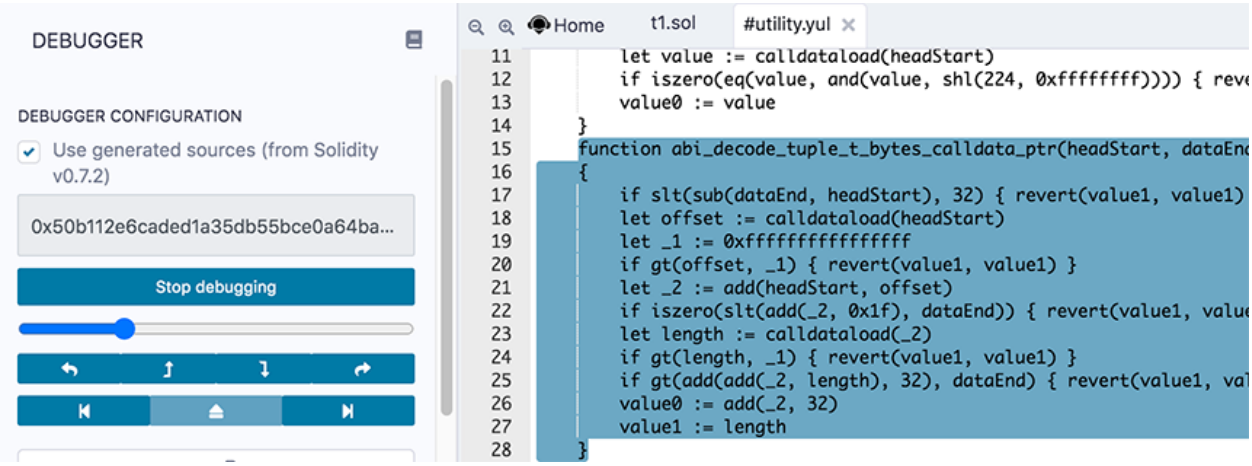
This page will go over the Debugger's *Use generated sources* option, its navigation and its panels.


DEBUGGER
DEBUGGER CONFIGURATION
Use generated sources (from Solidity v0.7.2)
0xcf91721a9a5811c0f81850e74bc991841d...
Stop debugging
Function Stack
Solidity Locals
Solidity State
015 REVERT
016 JUMPDEST
017 POP
018 CALLER
019 PUSH1 00
021 DUP1
022 PUSH2 0100
Step details
Stack
Memory

## 10.1 Use generated sources

This option is available for contracts using Solidity 0.7.2 or greater. See the solidity blog for more details about generated sourcess.

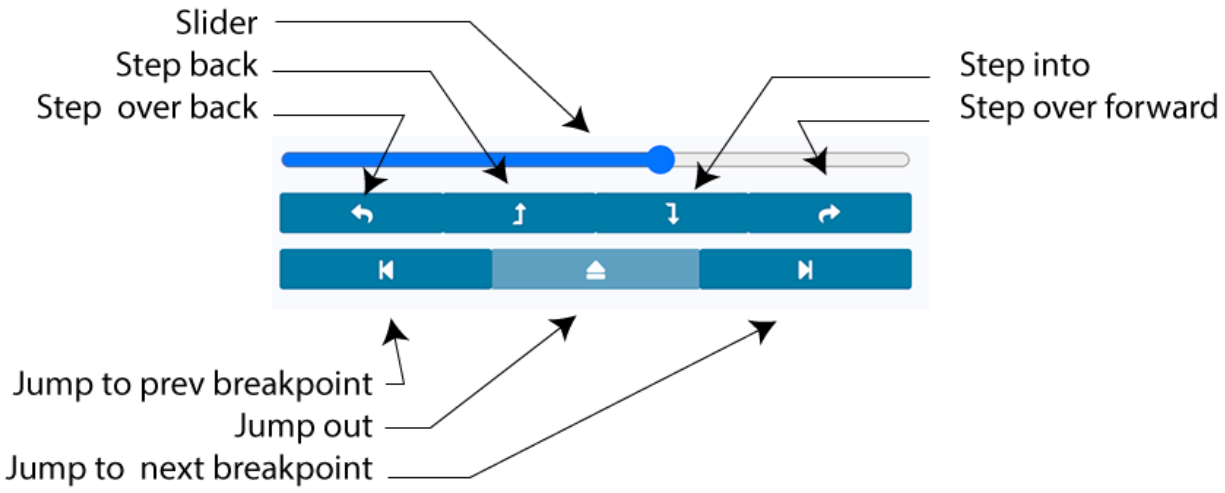
Using **generated sources** will make it easier to audit your contracts. When the option is checked, you can step into those compiler outputs — while debugging.




These compiler outputs will appear in a separate .yul file in the Remix editor.

## 10.2 The Debugger's Navigation

### 10.2.1 Slider & buttons




#### Slider

Moving the slider will highlight the relevant code in the **Editor**. On the most granular level, it scrolls through a transaction's opcodes (**see the opcode section below**). At each opcode, the transaction's state changes and these

changes are reflected in the **Debugger's panels**.

#### Step over back

This button goes to the previous opcode. If the previous step involves a **function call**, function will not be entered.

#### Step back

This button steps back to the previous opcode.

#### Step into

This button advances to the next opcode. If the next line contains a function call, **Step into** will go into the function.

#### Step over forward

This button advances to the next opcode. If the next step involves a **function call**, function will not be entered.

#### Jump to the previous breakpoint

Breakpoints can be placed in the gutter of the Editor. If the current step in the call has passed a breakpoint, this button will move the slider to the most recently passed breakpoint.

#### Jump out

When you are in a call and click on this button, the slider will be moved to the end of the call.

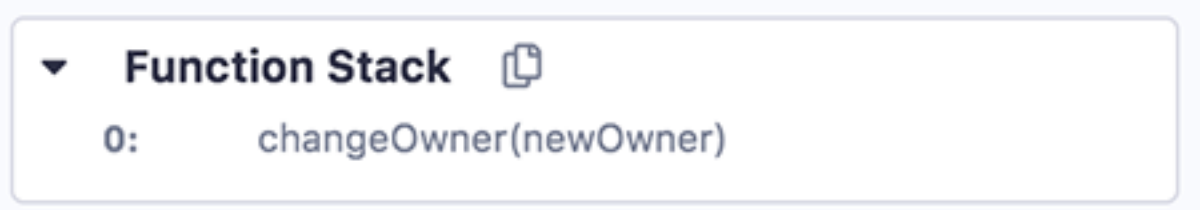
#### Jump to the next breakpoint

If a breakpoint is ahead in the code, this button will advance to that point.

## 10.3 The Debugger's Panels

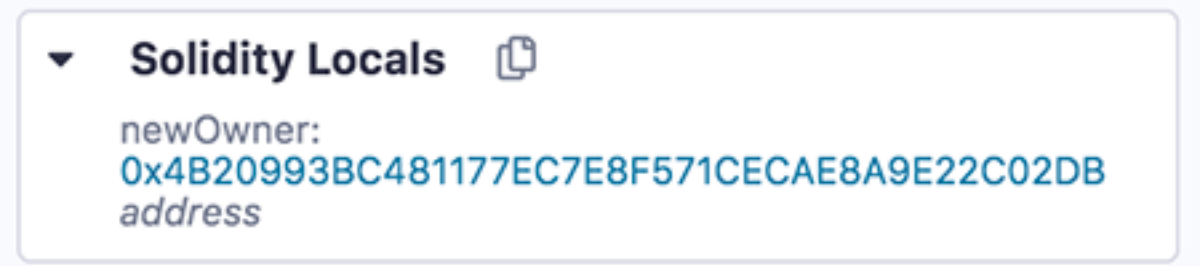
### 10.3.1 Function Stack

The Function stack lists the functions that the transaction is interacting with.



### 10.3.2 Solidity Locals

The Solidity Locals are the local variables inside a function.

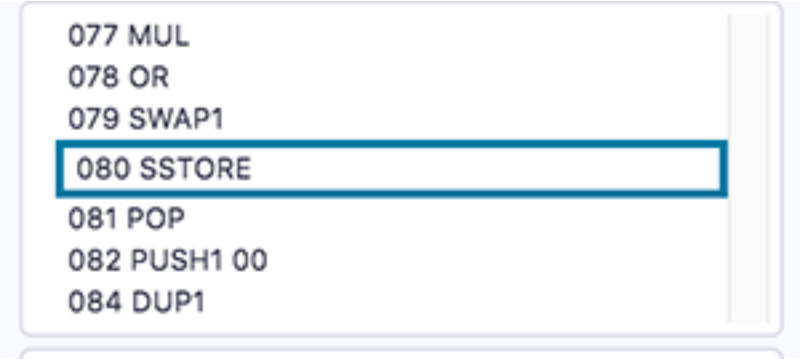



### 10.3.3 Solidity State

These are the state variables of the contract.


### 10.3.4 Opcodes

This panel shows the step number and the **opcode** that the debugger is currently on.



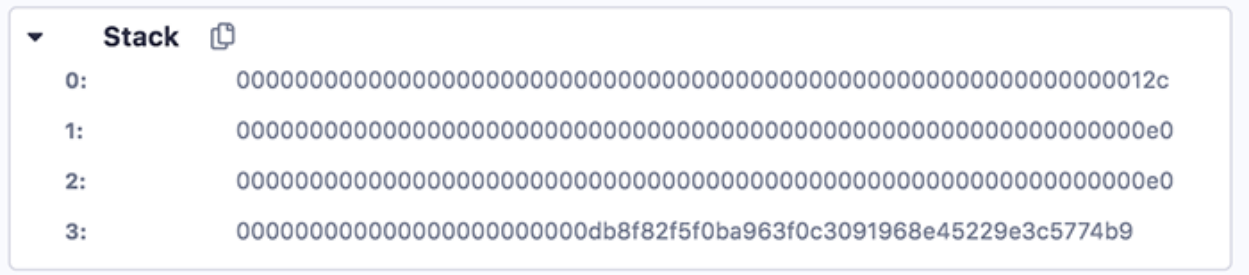

As you drag the **slider** (which is above the navigation buttons), the focussed step number & opcode changes.

### 10.3.5 Step details

Step details shows more info about the opcode step.


### 10.3.6 Stack

This panel shows the EVM Stack.




For more info about the stack.

### 10.3.7 Memory

Memory is cleared for each new message call. Memory is linear and can be addressed at byte level. **Reads** are limited to a width of 256 bits while **writes** can be either 8 bits or 256 bits wide.

The Memory panel consists of 3 columns. You might need to make Remix's side panel a bit wider to get the formatting to be correct. (Drag the border between the main panel and the side panel to the right).

The 1st column is the location in memory. The 2nd column is the hex encoded value. The 3rd column is the decoded value. If there is nothing, then the question marks (**?**) will show - like this:

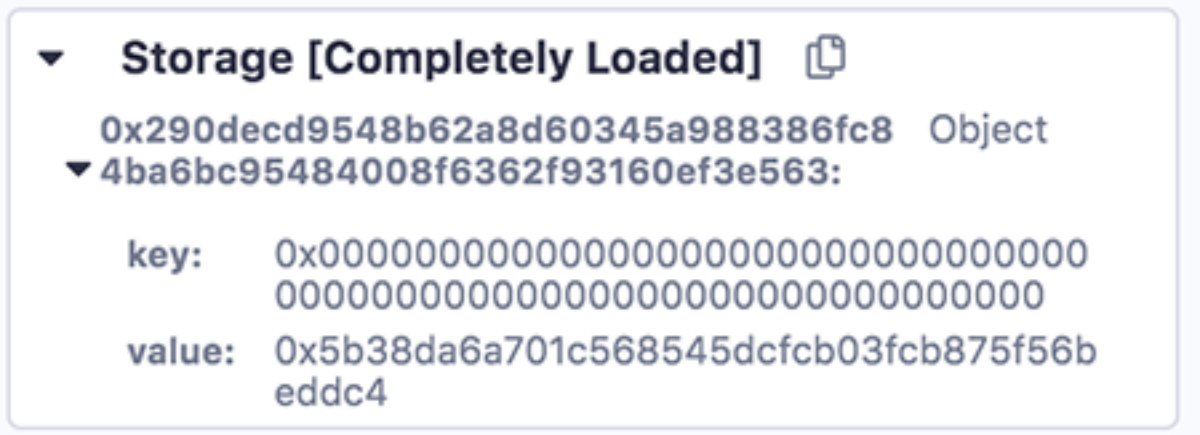
```
0x10: 00000000000000000000000000000000 ????????????????
```

Here is a full example of the **Memory** panel,


Some address slots have hex encoded values and those values are then decoded. For example, check position **0xa0** and **0x140**.

### 10.3.8 Storage

This is the persistant storage.




### 10.3.9 Call Stack

All computations are performed on a data array called the **call stack**. It has a maximum size of 1024 elements and contains words of 256 bits.


### 10.3.10 Call Data

The call data contains the functions parameters.


### 10.3.11 Return Value

The refers to the value that the function will return.

### 10.3.12 Full Storage Changes

This shows the persistant storage at the end of the function.


## 10.4 Breakpoints

Breakpoints can be placed in the gutter of the Editor to pause the debugger.

## 10.5 Additional Info

The debugger's granular information gives users detailed information about what is happening in a transaction - so not only is the debugger good for debugging, it is also an excellent teaching tool.

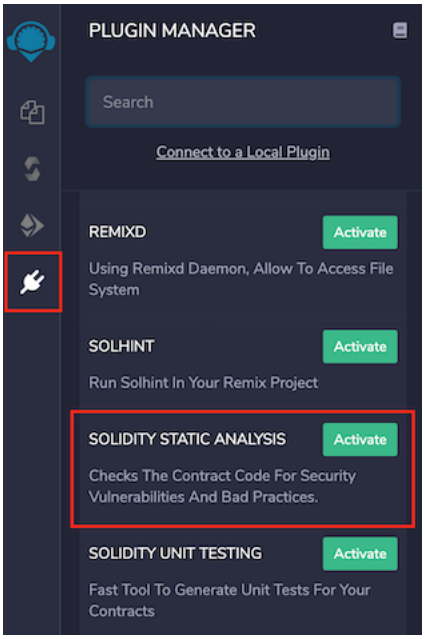
To learn about using the debugger, go to Debugging Transactions.

CHAPTER 11

# Solidity Static Analysis

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

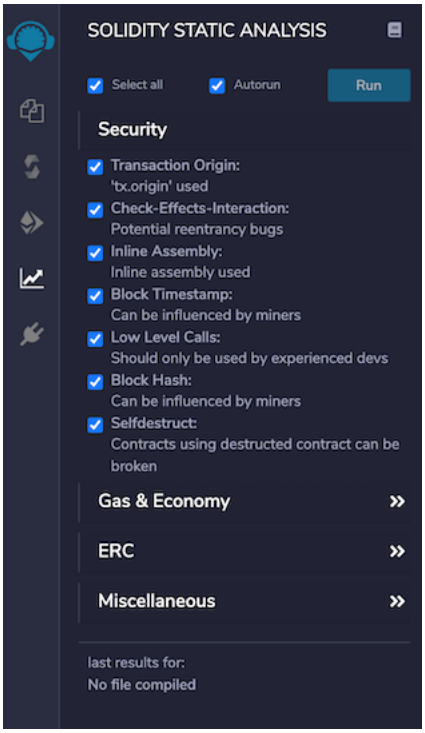
`Solidity Static Analysis` plugin performs static analysis on Solidity smart contracts once they are compiled. It checks for security vulnerabilities and bad development practices, among other issues. It can be activated from Remix `Plugin Manager`.




This plugin comes with `Solidity` environment of Remix IDE.

## 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. `Run` button will be disabled as there is no compiled contract for now.




By default, all modules are selected for analysing a smart contract.

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

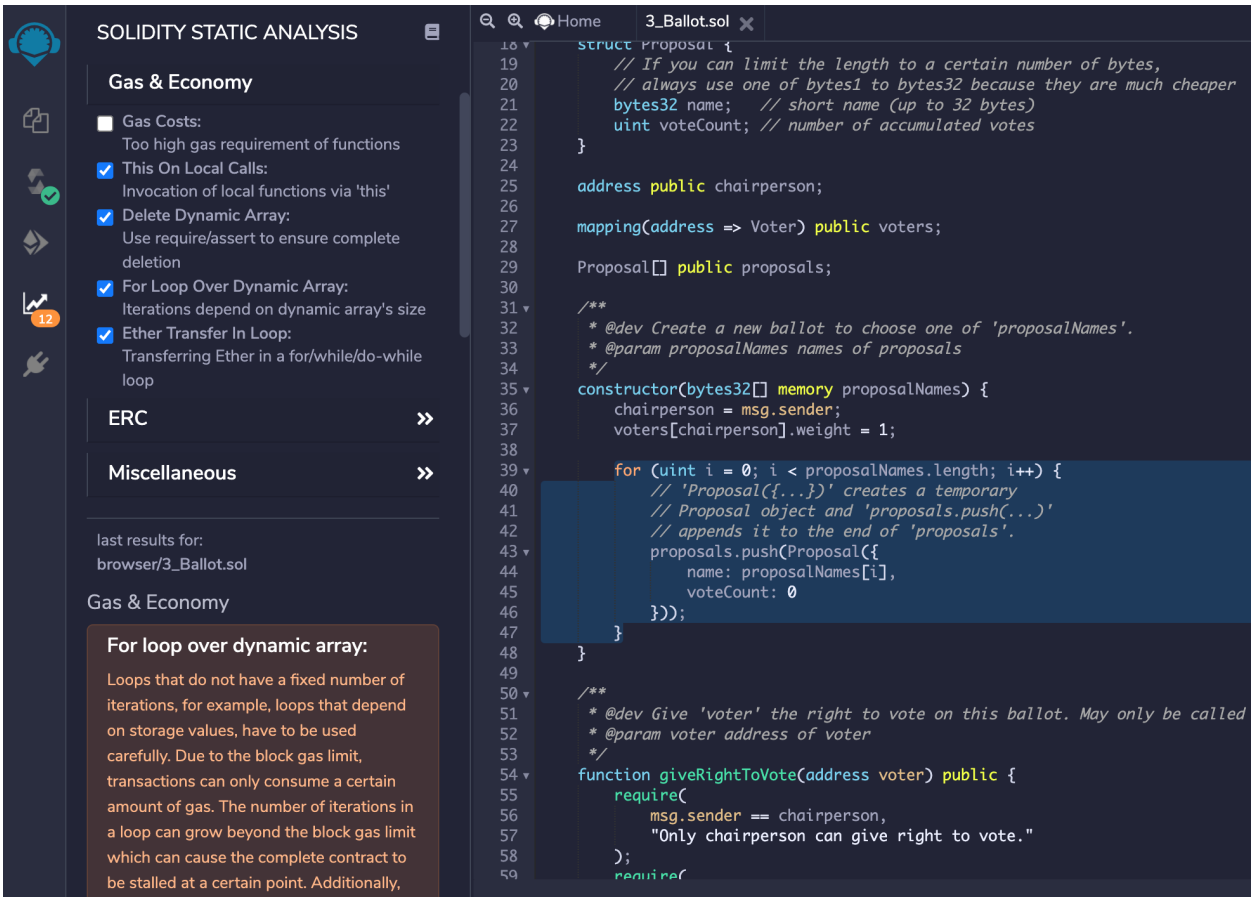
If `Auto run` checkbox is checked, analysis will be performed each time a contract is compiled. Uncheck the checkbox if you want to stop this behaviour.

## 11.2 Run

If `Auto run` checkbox is checked, analysis will be performed on compiling a contract and result will be shown as badge to the plugin icon. This number tells warnings count for the contract (e.g; `12` in attached image below) .

By visiting the plugin UI, the details of the warning can be seen along with the category name for each warning.

Clicking on warning details will highlight the relevant code in the editor.




## 11.3 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**:

### 11.3.1 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..));
```

### 11.3.2 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++;
        }
    }
}
```

### 11.3.3 Category: ERC

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

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

*Example:*

```
contract EIP20 {

    uint public decimals = 12;
}
```

### 11.3.4 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);
}
```

## 11.4 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

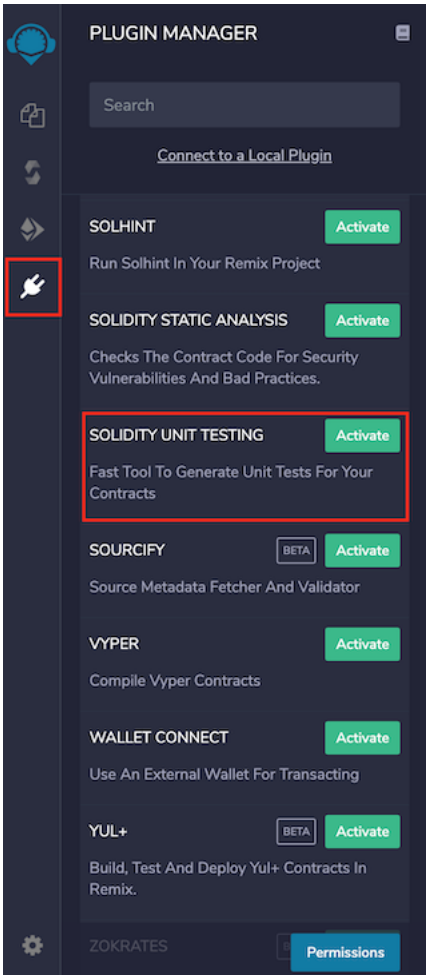
CHAPTER 12

# Unit Testing Plugin

Click the  (double check) icon from icon bar to move 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.

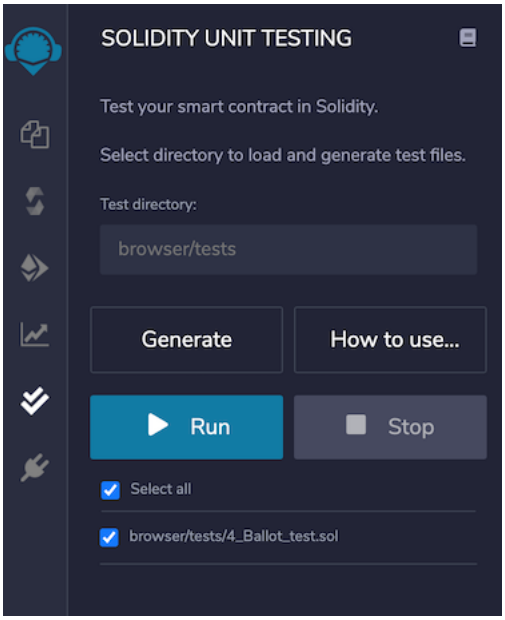
Go to the plugin manager by clicking the  (plug) icon and activate `Solidity Unit Testing` plugin.




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

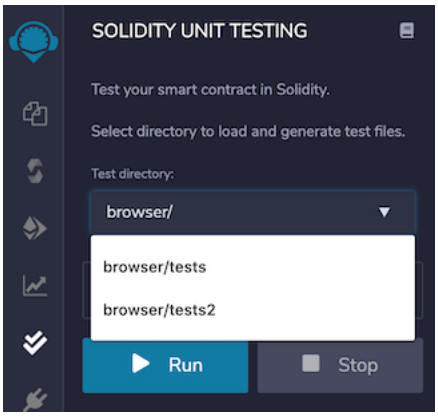
Alternatively, just select `Solidity` environment from Remix IDE `Home` tab. This will activate `Solidity Unit Testing` plugin along with `Solidity Compiler`, `Deploy & Run Transactions` & `Solidity Static Analysis` plugins.

After successful loading, plugin looks like this:




## 12.1 Test directory

Plugin asks you to provide a directory which will be your workspace only for this plugin. To select directory, as soon as you add `/` to the path, it shows the possible options.




Once selected, this directory will be used to load test files and to store newly generated test files.

Default test directory is `browser/tests`.

## 12.2 Generate

Select a solidity file which you want to test and click on the button `Generate`. It will generate a test file dedicated to selected file **in the test directory**.

If no file is selected, it will still create a file with generic name as `newFile_test.sol`.

This file contains sufficient information to give better understanding about developing tests for a contract.

Generic file looks as:

```
pragma solidity >=0.4.22 <0.8.0;
import "remix_tests.sol"; // this import is automatically injected by Remix.
import "remix_accounts.sol";
// Import here the file to test.

// File name has to end with '_test.sol', this file can contain more than one
↪testSuite contracts
contract testSuite {

    /// 'beforeAll' runs before all other tests
    /// More special functions are: 'beforeEach', 'beforeAll', 'afterEach' & 'afterAll
↪'
    function beforeAll() public {
        // Here should instantiate tested contract
        Assert.equal(uint(1), uint(1), "1 should be equal to 1");
    }

    function checkSuccess() public {
        // Use 'Assert' to test the contract,
        // See documentation: https://remix-ide.readthedocs.io/en/latest/assert_
↪library.html
        Assert.equal(uint(2), uint(2), "2 should be equal to 2");
        Assert.notEqual(uint(2), uint(3), "2 should not be equal to 3");
    }

    function checkSuccess2() public pure returns (bool) {
        // Use the return value (true or false) to test the contract
        return true;
    }

    function checkFailure() public {
        Assert.equal(uint(1), uint(2), "1 is not equal to 2");
    }

    /// Custom Transaction Context
    /// See more: https://remix-ide.readthedocs.io/en/latest/unittesting.html
↪#customization
    /// #sender: account-1
    /// #value: 100
    function checkSenderAndValue() public payable {
        // account index varies 0-9, value is in wei
        Assert.equal(msg.sender, TestsAccounts.getAccount(1), "Invalid sender");
        Assert.equal(msg.value, 100, "Invalid value");
    }
}
```

## 12.3 Write Tests

Write sufficient unit tests to ensure that your contract works as expected under different scenarios.

Remix injects a built-in `assert` library which can be used for testing. You can visit the library documentation *here*.

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

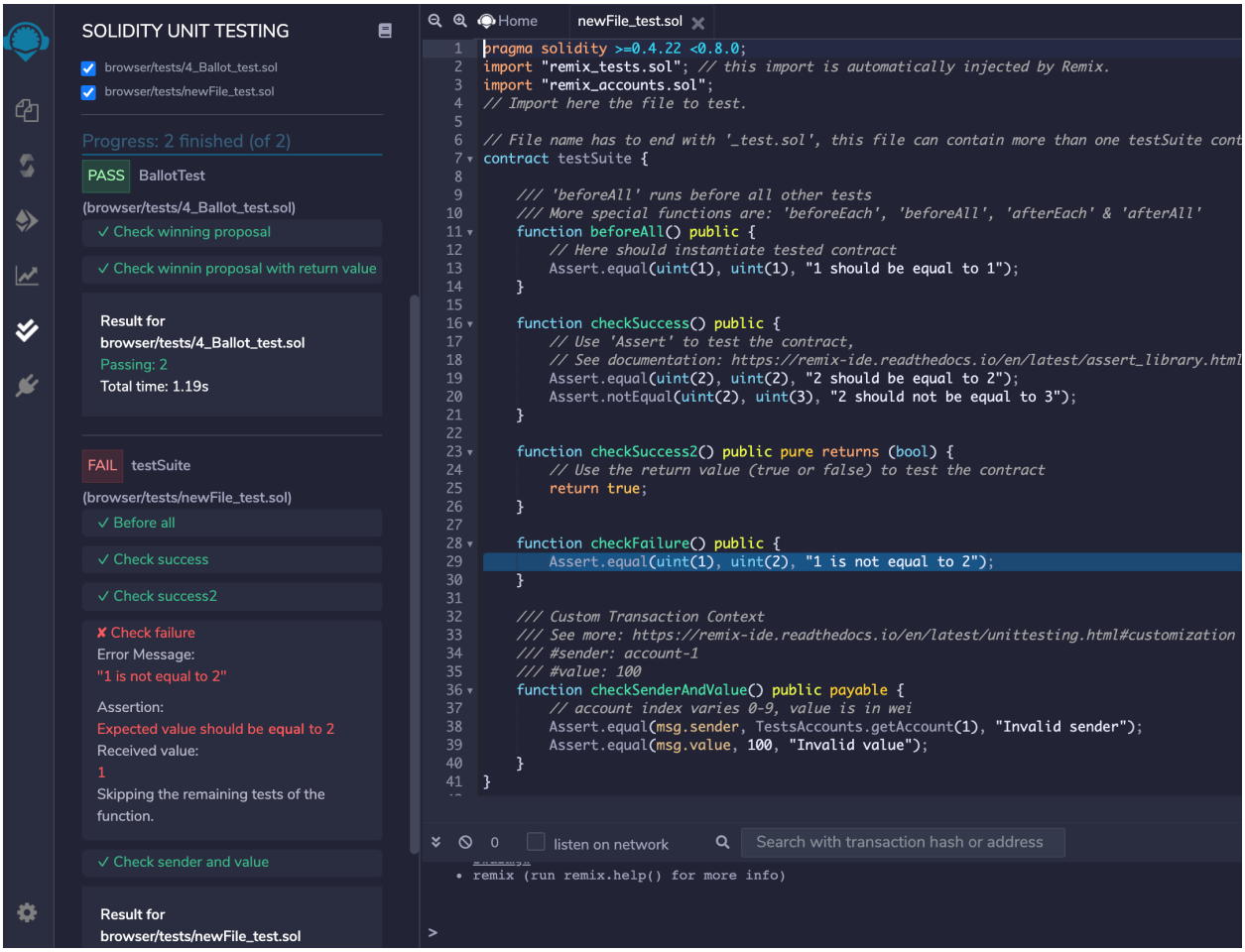
- `beforeEach()` - Runs before each test

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

To get started, see this simple example.

## 12.4 Run

Once you are done with writing tests, select the file(s) and click on `Run` to execute the tests. The execution will run in a separate environment. After completing the execution of one file, a test summary will be show as below:




For failed tests, there will be more assertion details to analyze the issue. Clicking on failed test will highlight the relevant line of code in the editor.

## 12.5 Stop

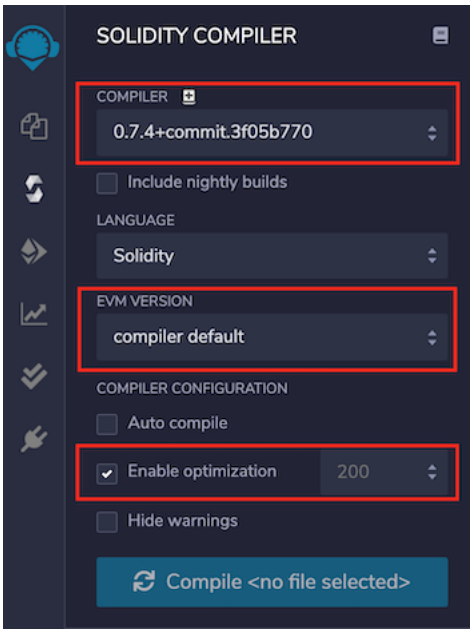
If you have selected multiple files to run the tests and want to stop the execution, click on `Stop` button. It will stop execution after running the tests for current file.

## 12.6 Customization

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

**1. Custom Compiler Context**

`Solidity Unit Testing` refers to the `Solidity Compiler` plugin for compiler configurations. Configure `Compiler`, `EVM Version`, `Enable Optimization` & `runs` in the `Solidity Compiler` plugin and this will be used in the `Solidity Unit Testing` plugin for contract compilation before running unit tests.




**2. Custom Transaction Context**

For interacting with a contract's method, the prime parameters of a transaction are `from` address, `value` & `gas`. Typically, a method's behaviour is tested with different values of these parameters.

One can input custom values for `msg.sender` & `msg.value` of transaction using NatSpec comments, 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");
}
```

Instructions to use:

1. Parameters must be defined in the method's NatSpec
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 only available for parameters `sender` & `value`
4. Sender is the `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.

## 12.7 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
- While a test file which imports `remix_accounts.sol` might not compile successfully with `Solidity Compiler` plugin, do not worry, this will have no bearing on its success with `Solidity Unit Testing` plugin.

# CHAPTER 13

# Command Line Interface

## 13.1 remix-tests

`remix-tests` is a tool which can be used as a CLI (Command Line Interface) solution to run the solidity unit tests. This is the same tool which works as a library underneath Remix's `Solidity Unit Testing` plugin. It is available on NPM as `@remix-project/remix-tests`.

## 13.2 Get started

You can install it using NPM:

- As a dev dependency:

```
npm install --save-dev @remix-project/remix-tests
```

- As a global NPM module:

```
npm install -g @remix-project/remix-tests
```

To confirm installation, run:

```
$ remix-tests version
0.1.36
```

Version should be same as on NPM.

## 13.3 How to use

You can see all available options using `help` command.

```
$ remix-tests help
Usage: remix-tests [options] [command]

Options:
  -V, --version            output the version number
  -c, --compiler <string>  set compiler version (e.g: 0.6.1, 0.7.1 etc)
  -e, --evm <string>       set EVM version (e.g: petersburg, istanbul etc)
  -o, --optimize <bool>    enable/disable optimization
  -r, --runs <number>      set runs (e.g: 150, 250 etc)
  -v, --verbose <level>    set verbosity level (0 to 5)
  -h, --help               output usage information

Commands:
  version                  output the version number
  help                     output usage information
```

General structure of a command is as:

`$ remix-tests <options> <file/directory path>`

To run all test files inside `examples` directory

```
$ remix-tests examples/
```

To run single test file named `simple_storage_test.sol` inside `examples` directory

```
$ remix-tests examples/simple_storage_test.sol
```

**NOTE:** `remix-tests` will assume that name of test(s) file ends with `"_test.sol"`. e.g `simple_storage_test.sol`

## 13.4 Example

Consider for a simple storage contract named `simple_storage.sol`:

```
pragma solidity >=0.4.22 <=0.8.0;

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 file `simple_storage_test.sol` can be as:

```
pragma solidity >=0.4.22 <=0.8.0;
import "remix_tests.sol"; // injected by remix-tests
import "./simple_storage.sol";

contract MyTest {
  SimpleStorage foo;

  function beforeAll() public {
    foo = new SimpleStorage();
  }

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

  function initialValueShouldNotBe200() public returns (bool) {
    return Assert.notEqual(foo.get(), 200, "initial value is not correct");
  }

  function shouldTriggerOneFail() public {
    Assert.equal(uint(1), uint(2), "uint test 1 fails");
    Assert.notEqual(uint(1), uint(2), "uint test 2 passes");
  }

  function shouldTriggerOnePass() public {
    Assert.equal(uint(1), uint(1), "uint test 3 passes");
  }
}
```

Running `simple_storage_test.sol` file will output as:

```
$ remix-tests simple_storage_test.sol

                :: Running remix-tests - Unit testing for solidity ::

'creation of library remix_tests.sol:Assert pending...'

          MyTest
        ✓  Initial value should be100
        ✓  Initial value should not be200
          Should trigger one fail
        ✓  Should trigger one pass


3 passing (0.282s)
1 failing

  1) MyTest: Should trigger one fail

         error: uint test 1 fails
         expected value to be equal to: 2
         returned: 1
```

## 13.5 Custom compiler context

Most of the `remix-tests` options are there to define a custom compiler context. With an extended custom compiler context, execution of above test file will go as:

```
$ remix-tests --compiler 0.7.4 --evm istanbul --optimize true --runs 300 simple_
↪storage_test.sol

                :: Running remix-tests - Unit testing for solidity ::

[14:03:18] info: Compiler version set to 0.7.4. Latest version is 0.8.0
[14:03:18] info: EVM set to istanbul
[14:03:18] info: Optimization is enabled
[14:03:18] info: Runs set to 300
Loading remote solc version v0.7.4+commit.3f05b770 ...
'creation of library remix_tests.sol:Assert pending...'

          MyTest
        ✓  Initial value should be100
        ✓  Initial value should not be200
          Should trigger one fail
        ✓  Should trigger one pass


3 passing (0.316s)
1 failing

  1) MyTest: Should trigger one fail

         error: uint test 1 fails
         expected value to be equal to: 2
         returned: 1
```

Rememeber, custom compiler version will require internet connection to load compiler.

## 13.6 As a CI solution

`remix-tests` can also be used for continuous integration (CI) testing.

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

CHAPTER 14

# Remix Assert Library



## 14.1 Assert

### 14.1.1 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
```

### 14.1.2 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 100");
// error: value should be 100
```

### 14.1.3 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
```

### 14.1.4 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
```

### 14.1.5 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 lesser than 1
```

# CHAPTER 15

# 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.

## 15.1 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");
    }
}
```

## 15.2 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');
    }
}
```

## 15.3 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');
    }
}
```

## 15.4 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');
    }
}
```

## 15.5 5. Testing a method involving `msg.sender` and `msg.value`

In the following test, we will be emulating multiple accounts making deposits in a smart contract to the same recipient and finally having the recipient withdraw the lump sum of all donations. We are also verifying that the donations match the expected amounts. This example really drives home how could you switch between different accounts, while using a set of different msg.value amounts.

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

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.4;

contract donations{
    struct Donation {
        uint id;
        uint amount;
        string donor;
        string message;
        uint timestamp; //seconds since unix start
    }
    uint amount = 0;
    uint id = 0;
    mapping(address => uint) public balances;
    mapping(address => Donation[]) public donationsMap;

    function donate(address _recipient, string memory _donor, string memory _msg)␣
↪public payable {
        require(msg.value > 0, "The donation needs to be >0 in order for it to go␣
↪through");
```

(continues on next page)

(continued from previous page)

```
        amount = msg.value;
        balances[_recipient] += amount;
        donationsMap[_recipient].push(Donation(id++,amount,_donor,_msg,block.
↪timestamp));
    }

    function withdraw() public {  //whole thing by default.
        amount = balances[msg.sender];
        balances[msg.sender] -= amount;
        require(amount > 0, "Your current balance is 0");
        (bool success,) = msg.sender.call{value:amount}("");
        if(!success){
            revert();
        }
    }

    function balances_getter(address _recipient) public view returns (uint){
            return balances[_recipient];
    }

    function getBalance() public view returns(uint) {
            return msg.sender.balance;
    }
}
```

Test contract/program: `Donations_test.sol`

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.4.22 <0.9.0;
import "remix_tests.sol";
import "remix_accounts.sol";
import "../donations.sol";

contract testSuite is donations {
    address acc0 = TestsAccounts.getAccount(0); //owner by default
    address acc1 = TestsAccounts.getAccount(1);
    address acc2 = TestsAccounts.getAccount(2);
    address acc3 = TestsAccounts.getAccount(3);
    address recipient = TestsAccounts.getAccount(4); //recipient

    /// #value: 1000000000000000000
    /// #sender: account-1
    function donateAcc1AndCheckBalance() public payable{
        Assert.equal(msg.value, 1000000000000000000, 'value should be 1 Eth');
        donate(recipient, "Mario", "Are you a bird?");
        Assert.equal(balances_getter(recipient), 1000000000000000000, 'balances␣
↪should be 1 Eth');
    }

    /// #value: 1000000000000000000
    /// #sender: account-2
    function donateAcc2AndCheckBalance() public payable{
        Assert.equal(msg.value, 1000000000000000000, 'value should be 1 Eth');
        donate(recipient, "Tom", "Are you a plane?");
        Assert.equal(balances_getter(recipient), 2000000000000000000, 'balances␣
↪should be 2 Eth');
```

(continues on next page)

(continued from previous page)

```
    }

    /// #value: 2000000000000000000
    /// #sender: account-3
    function donateAcc3AndCheckBalance() public payable{
        Assert.equal(msg.value, 2000000000000000000, 'value should be 1 Eth');
        donate(recipient, "Maria", "Are you a car?");
        Assert.equal(balances_getter(recipient), 4000000000000000000, 'balances␣
↪should be 4 Eth');
    }

    /// #sender: account-4
    function withdrawDonations() public payable{
        uint initialBal = getBalance();
        withdraw();
        uint finalBal = getBalance();
        Assert.equal(finalBal-initialBal, 4000000000000000000, 'balances should be 4␣
↪Eth');
    }
}
```

CHAPTER 16

# Hardhat Integration

*(Supported since Remix IDE v0.12.0 and Remixd v0.3.6)*

## 16.1 Remixd and Hardhat

**Note:** If you have not used `remixd` before, read more about it here

If `remixd` is running locally on your device and shared folder is a **Hardhat project**, an additional websocket plugin will be listening on port `65522`. According to its documentation,

> *Hardhat projects are npm projects with the hardhat package installed and a hardhat.config.js file.*

Remixd looks for the `hardhat.config.js` file in shared folder, and if it finds the file, the Hardhat websocket listener will run.

The Hardhat websocket listener is a websocket plugin similar to `remixd` and is used to perform Hardhat specific actions with Remix IDE.

It doesn't need any separate installation as it is shipped with remixd NPM module.

```
[INFO] you are using the latest version 0.3.5
[WARN] You can only connect to remixd from one of the supported origins.
[WARN] Any application that runs on your computer can potentially read from and write to all files in the directory.
[WARN] Symbolic links are not forwarded to Remix IDE

[INFO] Fri Jun 04 2021 10:47:44 GMT+0530 (India Standard Time) remixd is listening on 127.0.0.1:65520
[INFO] Fri Jun 04 2021 10:47:44 GMT+0530 (India Standard Time) hardhat is listening on 127.0.0.1:65522
```

## 16.2 Enable Hardhat Compilation

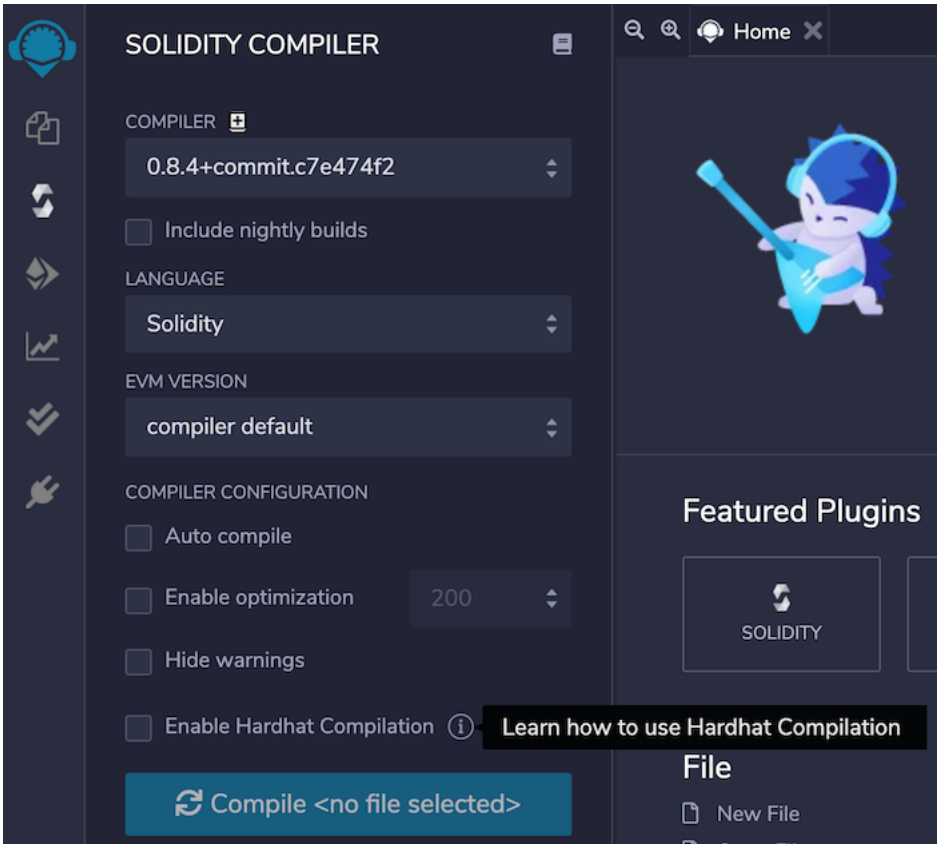
### 16.2.1 Prerequisites

To use Hardhat compilation with Remix IDE efficiently:

1. **Hardhat** should be installed locally on the system https://hardhat.org/getting-started/#installation
2. Shared folder should be a Hardhat project containing `hardhat.config.js`
3. `Remixd` Hardhat websocket listener should be running at `65522`

### 16.2.2 How to use

If a hardhat project is shared through remixd and `localhost` workspace is loaded in Remix IDE, there will be an extra checkbox shown in `Solidity Compiler` plugin with the label `Enable Hardhat Compilation`.

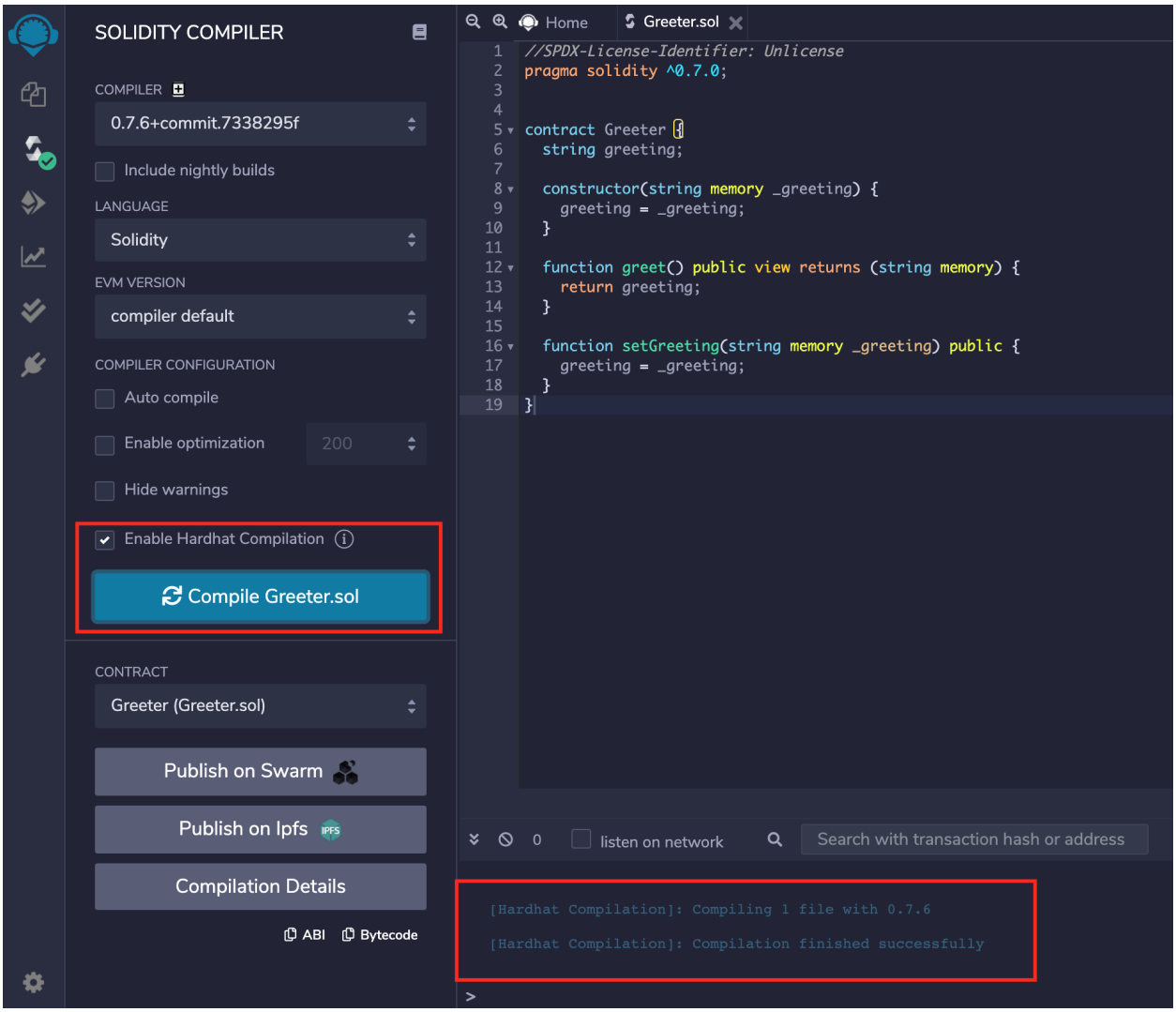



There is an info icon along side the label which redirects to a specific section of Remix official documentation that explains how to use Hardhat compilation.

One can check the `Enable Hardhat Compilation` box to run the compilation for Hardhat along with the Remix using the compiler configuration in `Solidity Compiler` plugin.

On clicking `Compile` button, a file with `remix-compiler.config.js` will be created on the project root which will be storing compiler configuration set in Remix's `Solidity Compiler` plugin. It is passed to Hardhat for compilation.

The result of the compilation will be shown in the Remix IDE terminal



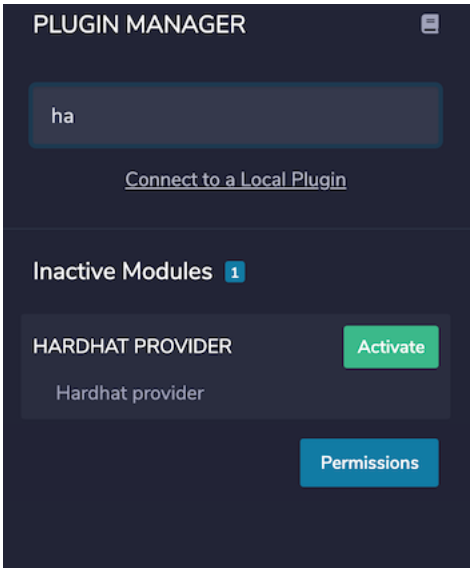

and also in the **remixd** terminal.

```
[Hardhat Compilation]: Compiling 1 file with 0.7.6

[Hardhat Compilation]: Compilation finished successfully
```

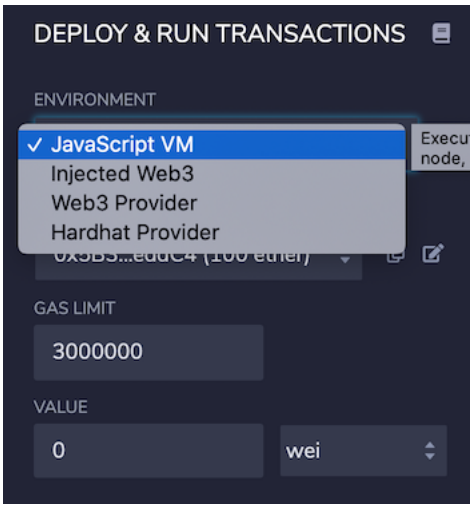
## 16.3 Hardhat Provider

*In Hardhat, contracts are deployed by starting a local node. Read more about it in Hardhat documentation*

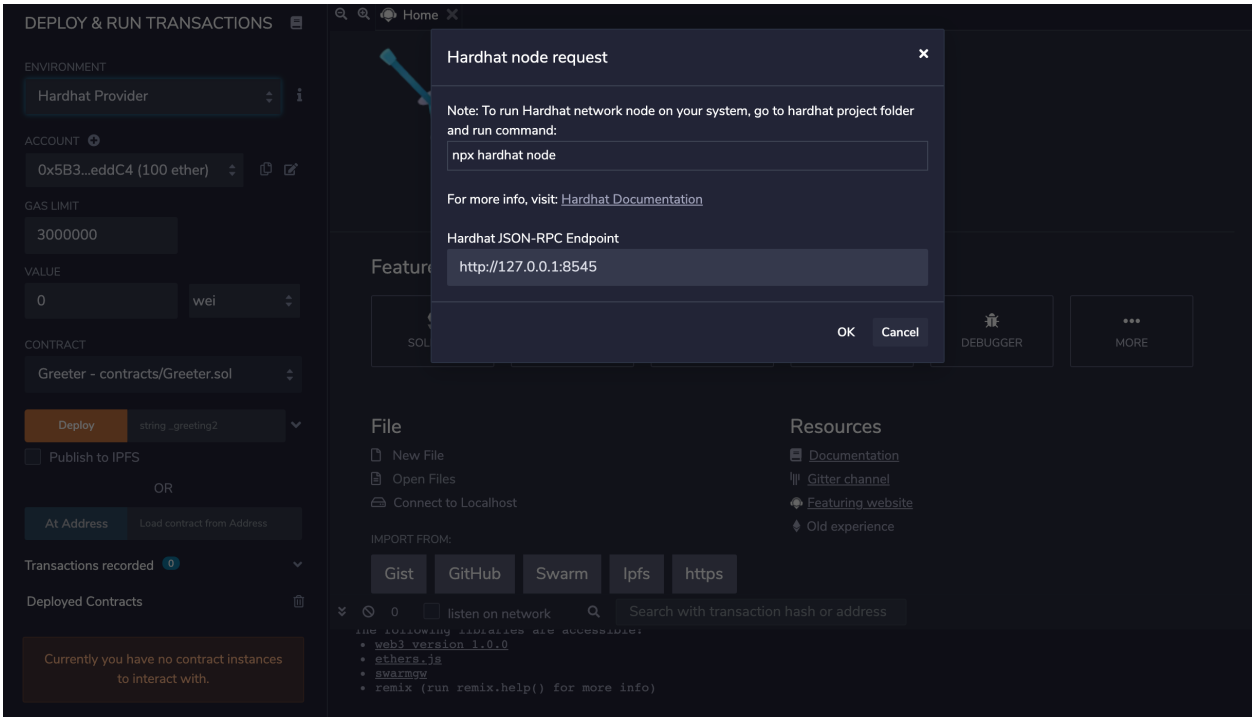
**Hardhat Provider** is a plugin on Remix IDE which enables users to deploy the contract to the Hardhat 'localhost' network. This plugin can be activated from Remix IDE plugin manager.




When activated, you will have an extra option in the `ENVIRONMENT` dropdown of `Deploy and Run Transactions` plugin with the label `Hardhat Provider`

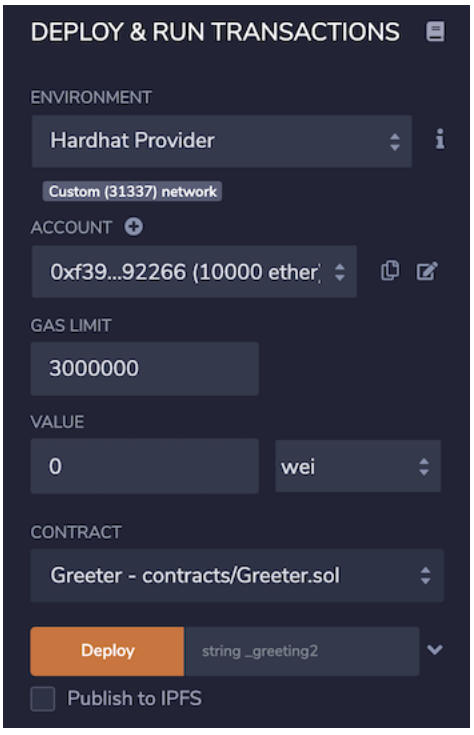



As soon as you select `Hardhat Provider`, a modal is opened asking for the `Hardhat JSON-RPC Endpoint`.




If Hardhat node is running with default options, then the default endpoint value in modal will not need any change. In case, Hardhat node host and port are different, JSON-RPC endpoint should be updated in the modal text box.

Once the correct endpoint is filled in the modal, just click on `OK` and the accounts from the Hardhat node will be loaded in the `ACCOUNT` section. Network id will also be shown.




Now, one can start deploying the contract from Remix IDE to the Hardhat local node as usual.

# CHAPTER 17

# Slither Integration

*(Supported since Remix IDE v0.15.0 and Remixd v0.5.0)*

## 17.1 Remixd and Slither

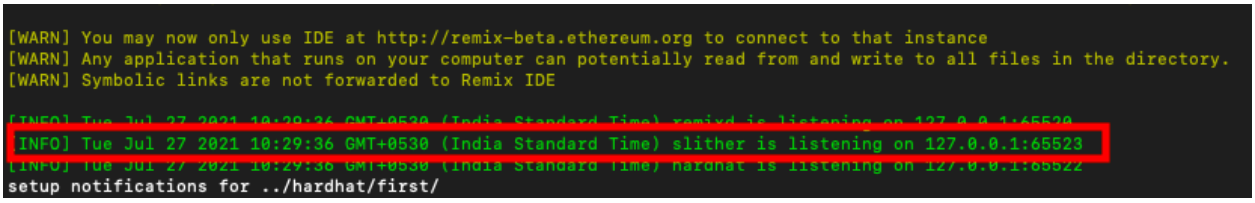
**Note:** If you have not used `remixd` before, read more about it here

Since Remixd `v0.5.0`, when `remixd` is running locally on your device, an additional websocket plugin will be listening on port `65523` which will be dedicated for Slither integration.

> *Slither is a Solidity static analysis framework written in Python 3. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses.*

The remixd Slither listener is a websocket plugin similar to remixd and is used to perform Slither analysis with Remix IDE.

It doesn't need any separate installation as it is shipped with remixd NPM module.




## 17.2 Enable Slither Analysis

### 17.2.1 Prerequisites

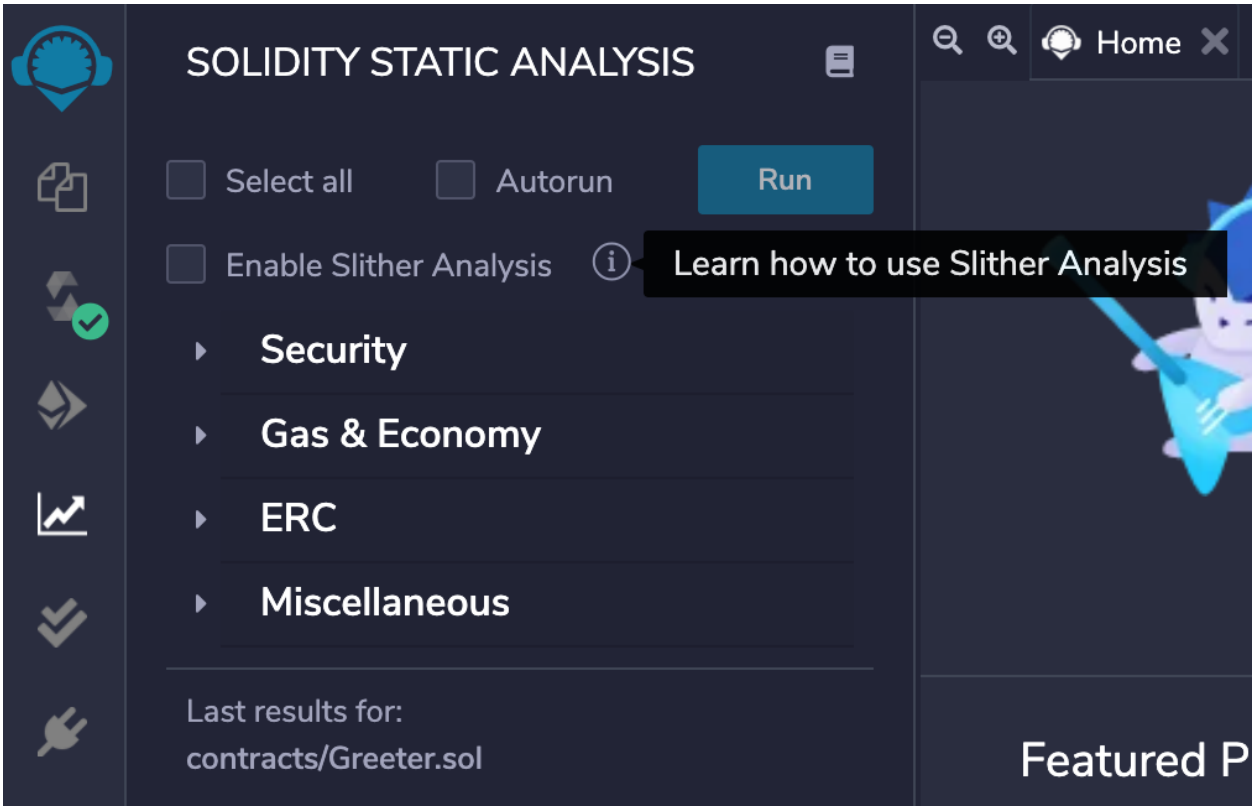
To use Slither analysis with Remix IDE efficiently, following tools should be installed locally on the system:

1. **Slither:** https://github.com/crytic/slither#how-to-install

2. **Solc:** https://docs.soliditylang.org/en/latest/installing-solidity.html
3. **Solc-select:** https://github.com/crytic/solc-select#quickstart

### 17.2.2 How to use

If a project is shared through remixd and `localhost` workspace is loaded in Remix IDE, there will be an extra checkbox shown in `Solidity Static Analysis` plugin with the label `Enable Slither Analysis`.

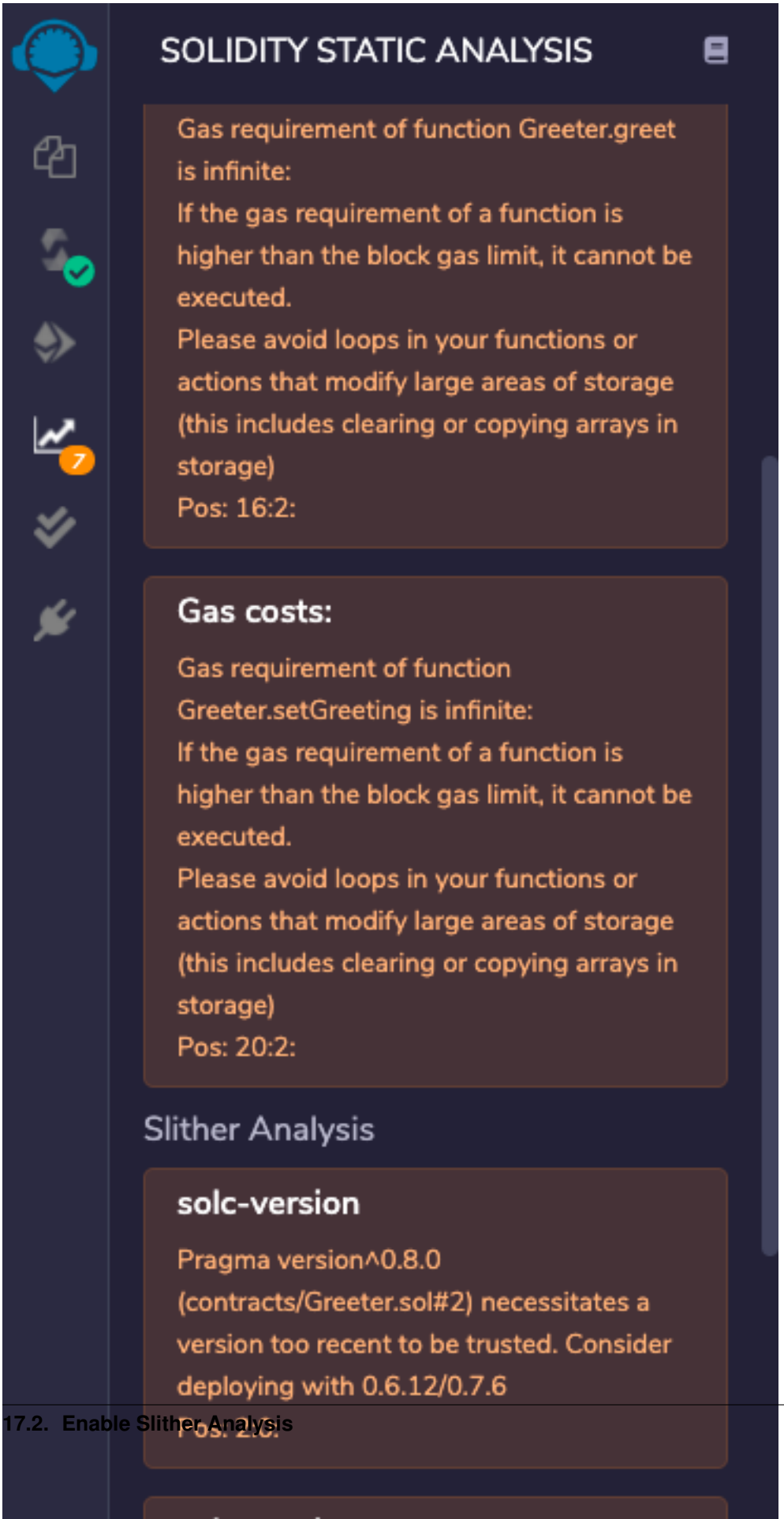



There is an info icon on the right side of the label which redirects to a specific section of Remix official documentation that explains how to use Slither Analysis and prerequisites for it.

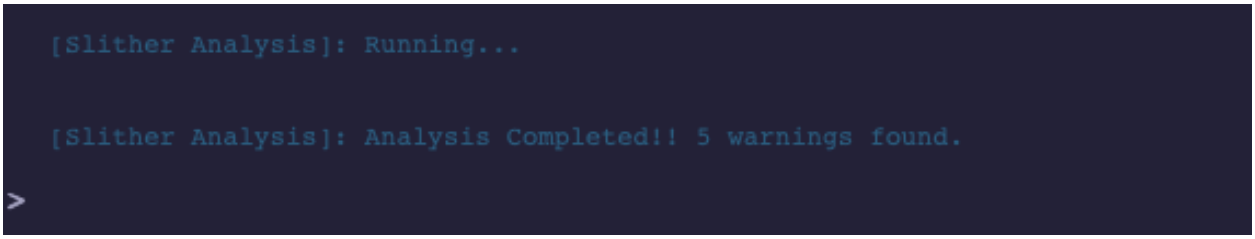
One can check the `Enable Slither Analysis` box to run the analysis using Slither along with the Remix's analysis library.

Generated report of Slither analysis will be stored locally on project root with a file name prefixed with `remix-slitherReport_`, for example: `remix-slitherReport_1627362090.json`.

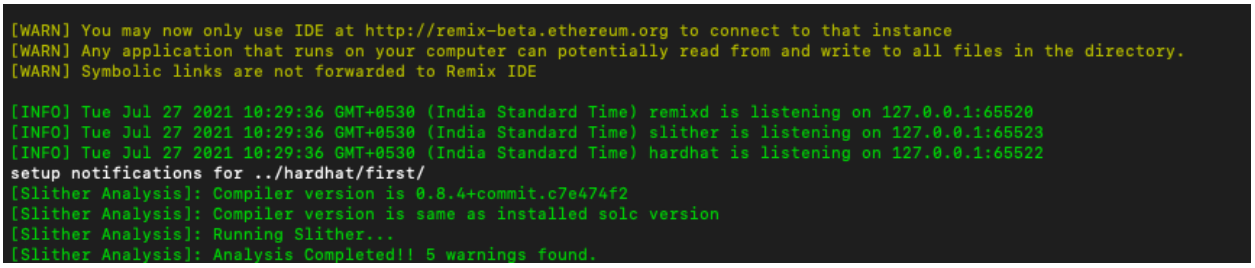
Slither Analysis report will also be displayed on the Remix IDE side after the Remix analysis report for better user readability.


SOLIDITY STATIC ANALYSIS
Gas requirement of function Greeter.greet is infinite:
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 (this includes clearing or copying arrays in storage)
Pos: 16:2:
Gas costs:
Gas requirement of function Greeter.setGreeting is infinite:
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 (this includes clearing or copying arrays in storage)
Pos: 20:2:
Slither Analysis
solc-version
Pragma version^0.8.0 (contracts/Greeter.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

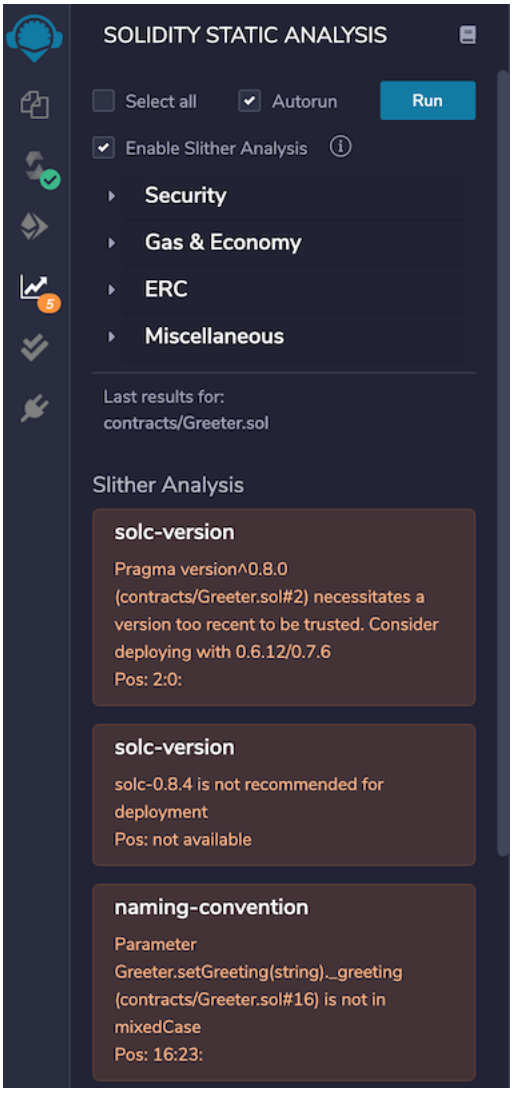
The result of the analysis will be shown in the Remix IDE terminal




and also in the **remixd** console.




To only run Slither Analysis, deselect `Select all` checkbox and click on `Run`. Now it will show only the Slither Analysis report.

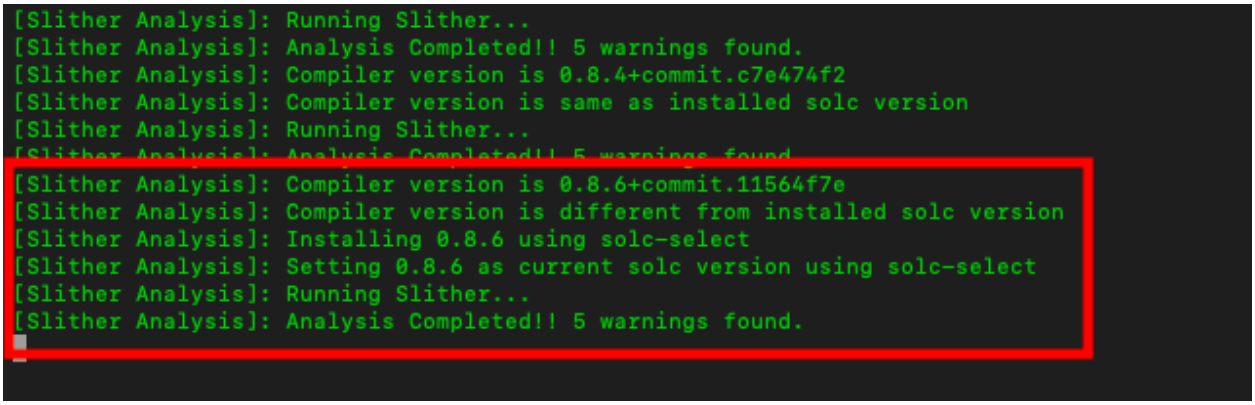



### 17.2.3 More Details

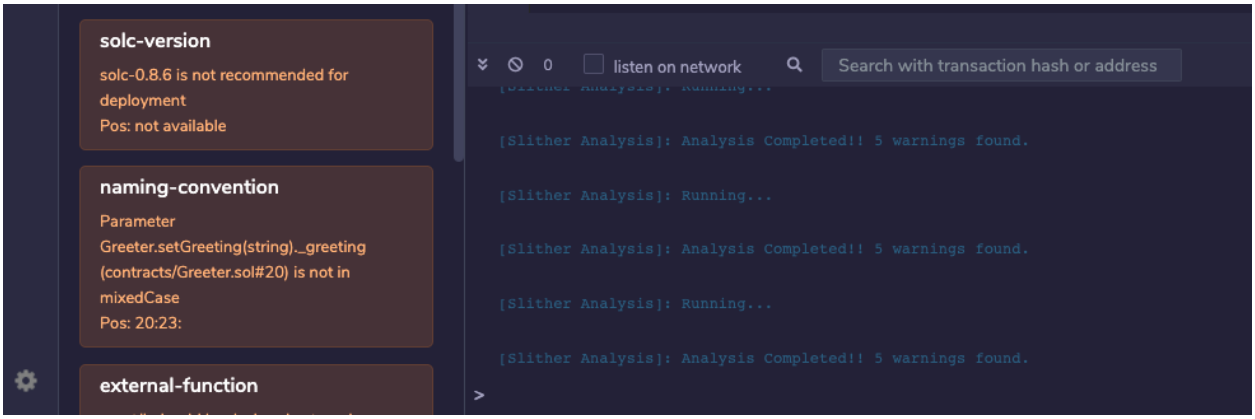
Analysis for Slither is run using the version set in `Solidity Compiler` plugin on Remix IDE. Slither is a CLI tool and requires `solc` to run the analysis. Before running the analysis, Slither websocket plugin checks if current version of solc is same as the version set in Remix IDE.

If the compiler version set in Solidity Compiler is different from current version of solc installed locally, the Slither websocket plugin will update the `solc` to be the same as the required version `solc-select`.

For example, if current `solc` version on the system is 0.8.4 and on the Remix IDE 0.8.6 is set, `remixd` logs explain remote solc version selection




After successful analysis run:



# CHAPTER 18

# Hardhat console.log Integration

*(Supported since Remix IDE v0.17.0)*

## 18.1 Prologue

*Hardhat Network allows you to print logging messages and contract variables by calling console.log() from your Solidity code. To use it, you simply import* `hardhat/console.sol` *and call it.*

For more: https://hardhat.org/hardhat-network/reference/#console-log

## 18.2 `console.log` in Remix IDE

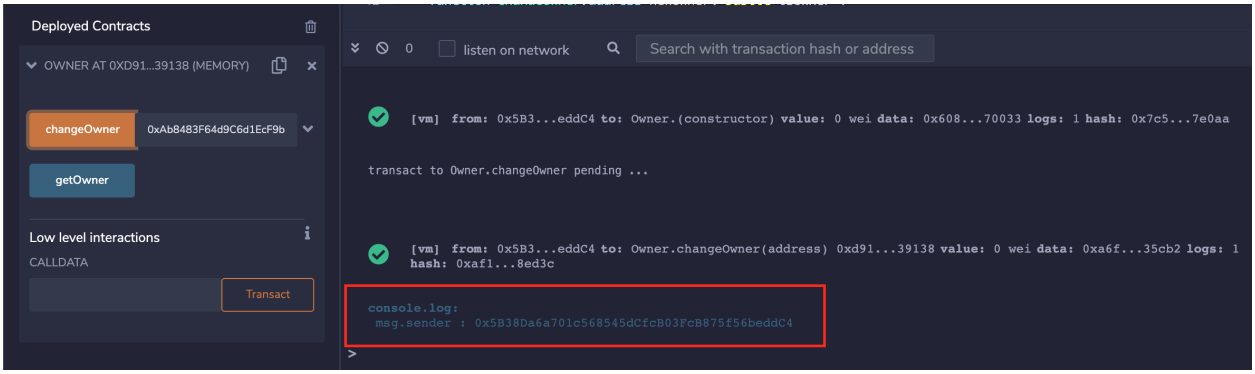
Remix IDE supports hardhat console library while using `JavaScript VM`. It can be used while making a transaction or running unit tests.

### 18.2.1 Deploy and Run Transactions

To try it out, you need to put an import statement and use `console.log` to print the value as shown in image.

```
Home  2_Owner.sol
 * @dev Set & change owner
 */

import "hardhat/console.sol";

contract Owner {

    address private owner;

    // event for EVM logging
    event OwnerSet(address indexed oldOwner, address indexed newOwner);

    // modifier to check if caller is owner
    modifier isOwner() {
        // If the first argument of 'require' evaluates to 'false', execution terminates and all
        // changes to the state and to Ether balances are reverted.
        // This used to consume all gas in old EVM versions, but not anymore.
        // It is often a good idea to use 'require' to check if functions are called correctly.
        // As a second argument, you can also provide an explanation about what went wrong.
        require(msg.sender == owner, "Caller is not owner");
        _;
    }

    /**
     * @dev Set contract deployer as owner
     */
    constructor() {
        owner = msg.sender; // 'msg.sender' is sender of current call, contract deployer for a constructor
        emit OwnerSet(address(0), owner);
    }

    /**
     * @dev Change owner
     * @param newOwner address of new owner
     */
    function changeOwner(address newOwner) public isOwner {
        console.log('msg.sender :', msg.sender);
        emit OwnerSet(owner, newOwner);
        owner = newOwner;
    }
```

Further, once you execute the `changeOwner` method, value from console statement will be shown in Remix terminal after transaction details as below:

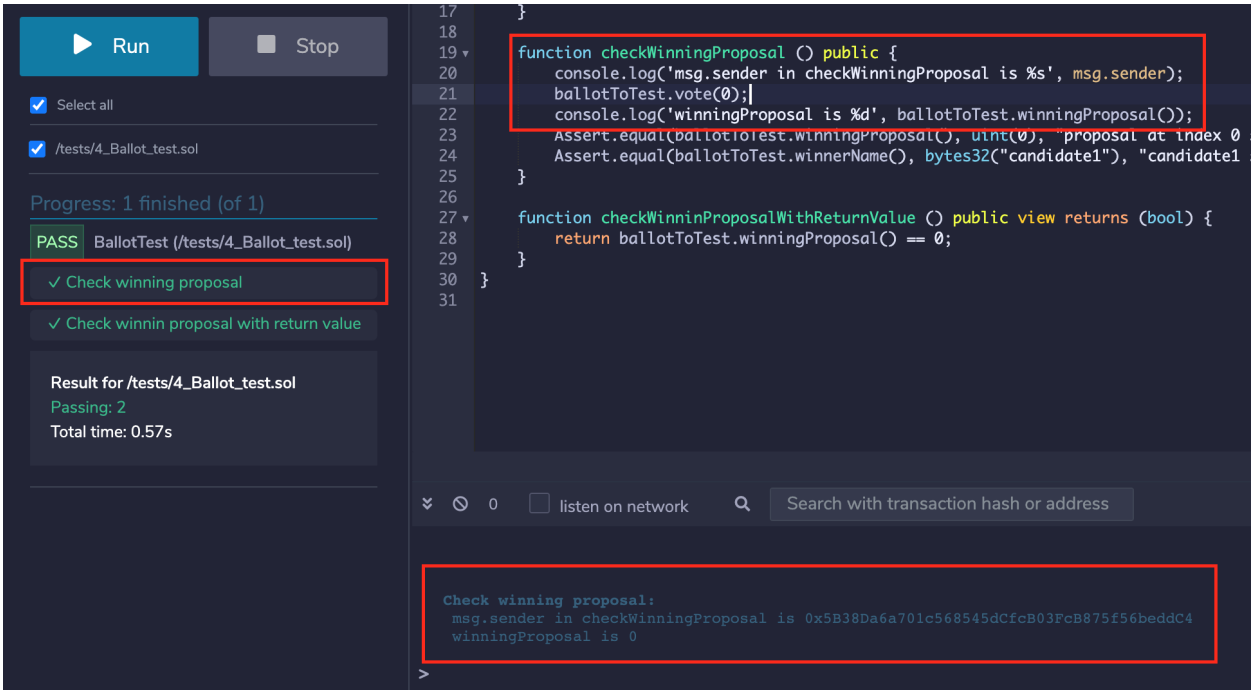



## 18.2.2 Solidity Unit Testing

Similarly, `console.log` can be used while running unit tests using Remix Solidity Unit Testing plugin. See image below.

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;
import "remix_tests.sol"; // this import is automatically injected by Remix.
import "../contracts/3_Ballot.sol";

import "hardhat/console.sol";

contract BallotTest {

    bytes32[] proposalNames;

    Ballot ballotToTest;
    function beforeAll () public {
        proposalNames.push(bytes32("candidate1"));
        ballotToTest = new Ballot(proposalNames);
    }

    function checkWinningProposal () public {
        console.log('msg.sender in checkWinningProposal is %s', msg.sender);
        ballotToTest.vote(0);
        console.log('winningProposal is %d', ballotToTest.winningProposal());
        Assert.equal(ballotToTest.winningProposal(), uint(0), "proposal at index 0 should be the winning proposal");
        Assert.equal(ballotToTest.winnerName(), bytes32("candidate1"), "candidate1 should be the winner name");
    }
```

For the tests including logging message, it will display in the Remix Terminal corresponding to test name.



# CHAPTER 19

# 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.

## 19.1 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"
                    }
            ]

}
```

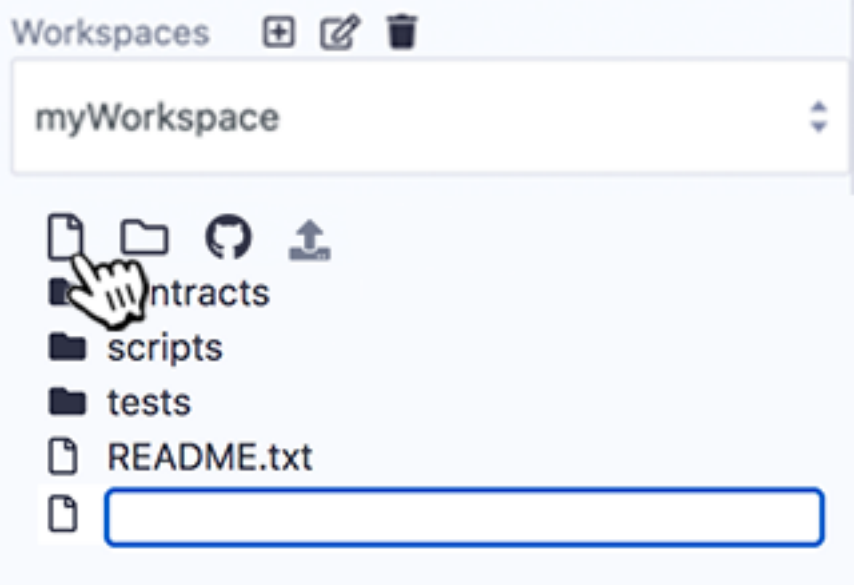
CHAPTER 20

# Creating and Deploying a Contract

This page contains the process of creating a contract, compiling it, deploying and then interacting with it.

## 20.1 A sample contract

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




Go to the File Explorer, create a new file, name it and in the editor paste the contract below.

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.7.6;
```

(continues on next page)

(continued from previous page)

```
contract testContract {

    uint256 value;

    constructor (uint256 _p) {
        value = _p;
    }

    function setP(uint256 _n) payable public {
        value = _n;
    }

    function setNP(uint256 _n) public {
        value = _n;
    }

    function get () view public returns (uint256) {
        return value;
    }
}
```

## 20.2 Compile the Contract

With the contract above as the active tab in the Editor, compile the contract.

**For More Info** see the docs on the (Solidity Compiler).

## 20.3 Deploy the contract

Go to the **Deploy & Run Transactions** plugin.

There are 3 type of environments Remix can be plugged to:

- Javascript VM
- Injected Web3
- Web3 Provider

(For details see Running transactions)

Both **Injected Web3** and **Web3 Provider** require the use of an external tool.

An external tool for **Injected provider** is Metamask. Some external tools used with **Web3 provider** are a Truffle Ganache-CLI, Hardhat node, or an Ethereum node itself.

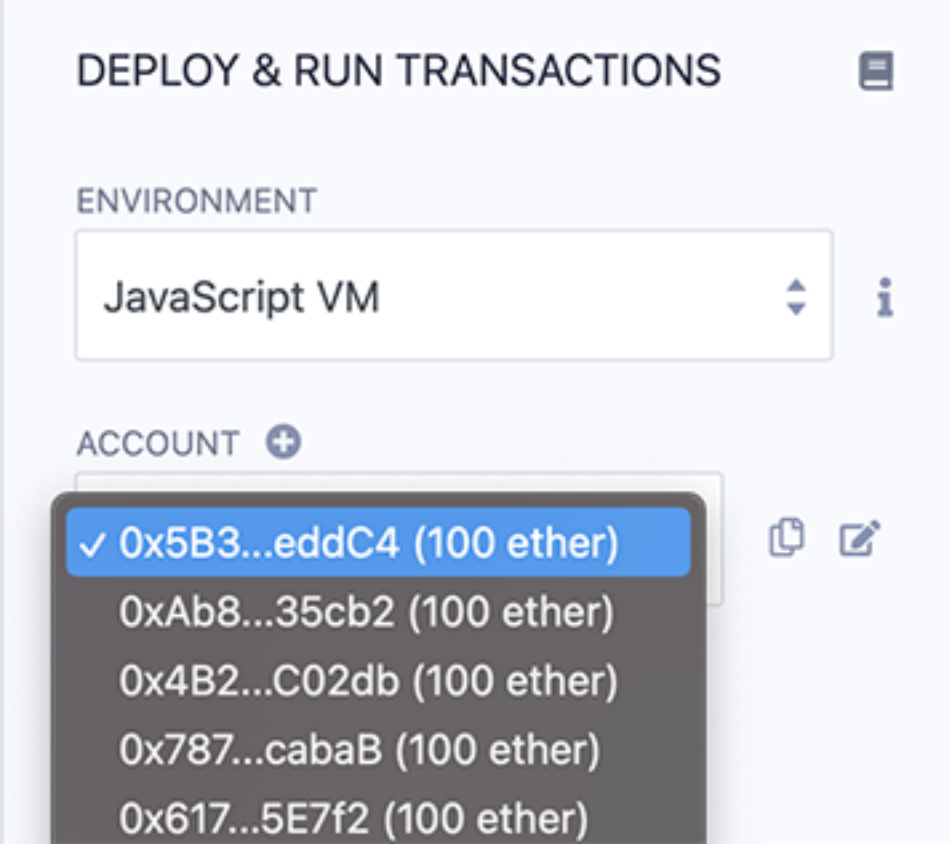
The **JavaScript VM** is convenient because it is a blockchain that runs in your browser and you don't need any other software or Ethereum node to run it.

**NOTE:** When you are in the **Javascript VM** and you reload the browser - the **Javascript VM** will restart to its fresh & default state.

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

## 20.4 Select the VM environment

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




## 20.5 Deploying a contract

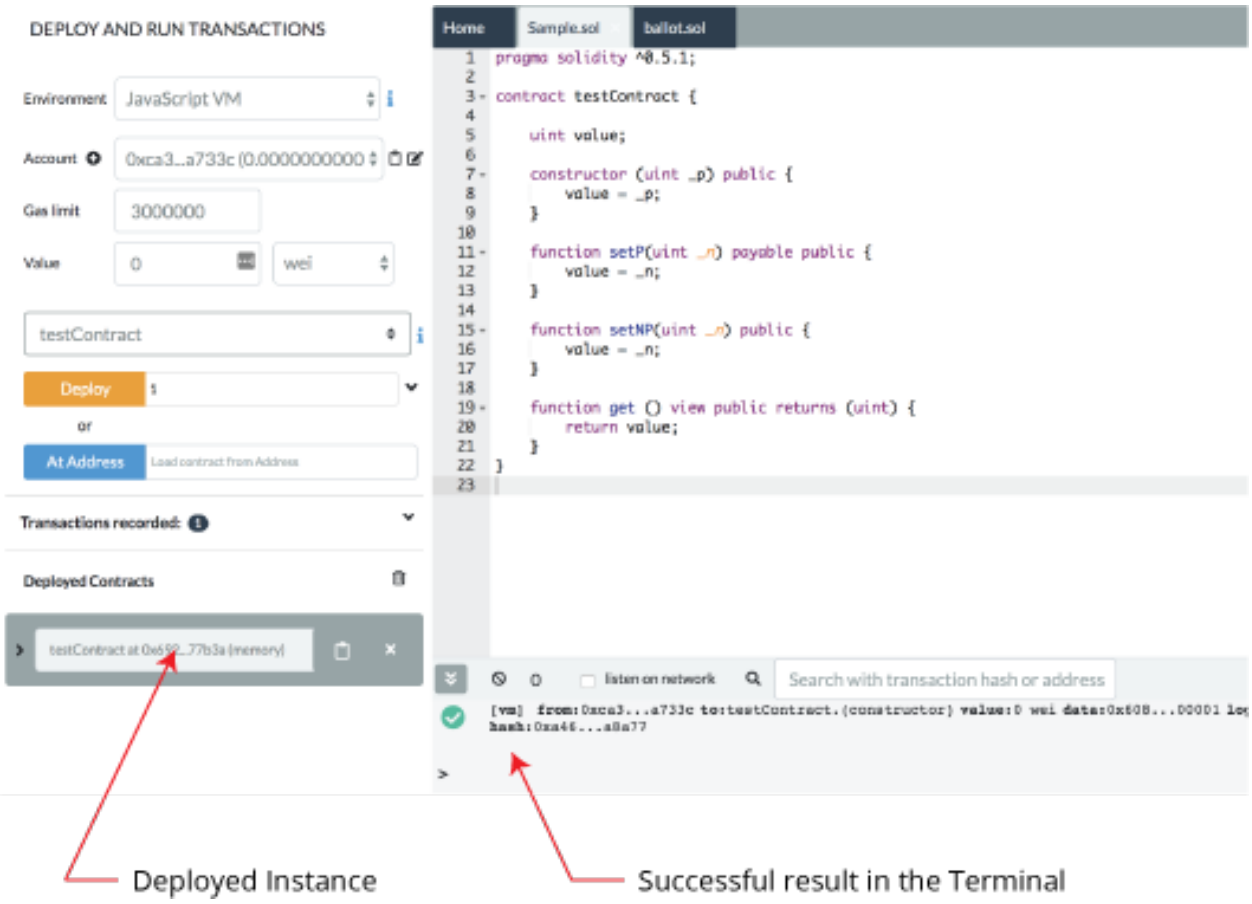

The constructor of `testContract` needs a parameter (of type `uint256`). Input a uint256 and click on `Deploy`.

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

In a "normal" blockchain, you would have to wait for the transaction to be mined. However, because we are using the `JavaScript VM`, our execution is immediate.

The terminal will give information about the transaction.

The newly created instance is displayed in the **Deployed Contracts** section.




## 20.6 Interacting with an instance

Clicking on the caret to the left of the instance of TESTCONTRACT will open it up so you can see its function.

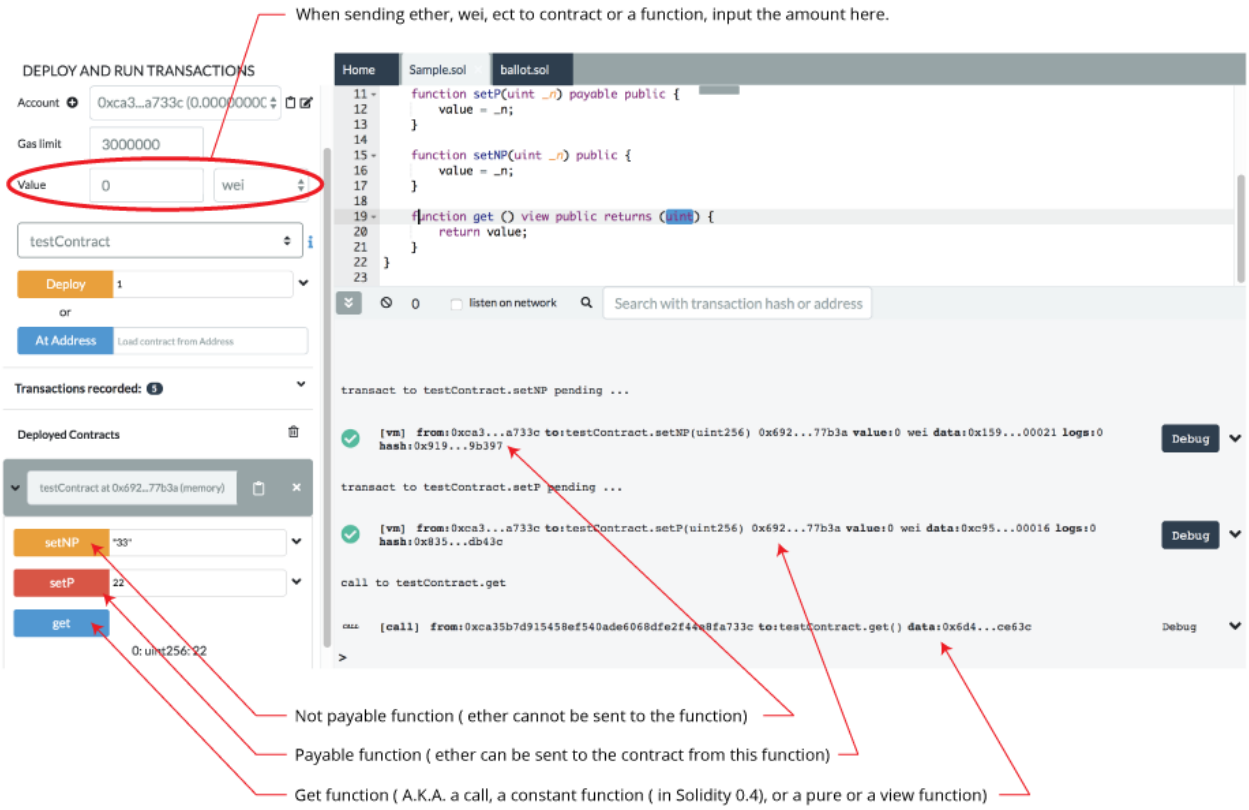
This new instance contains 3 actions which corresponds to the 3 functions (`setP`, `setPN`, `get`). Clicking on `setP` or `setPN` will create a new transaction.

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 (the variable `value`) of this instance.

Because `get` is a **view function**, you can see the return value just below the `get` button.


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: 5
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 21

# Debugging Transactions

(also see this page's companion: the Debugger Tour)

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

- Use Case 1: for debugging a transaction made in Remix - click the **Debug button** in the transaction log in Remix's Terminal.
- Use Case 2: for debugging a transaction where you have a **txn hash** from **verified contract** or where you have the txn hash and the compiled source code with the same compilation settings as the deployed contract.

## 21.1 Initiate Debugging from the transaction log in the Terminal

Let's start with a basic contract ( or replace the contract below with your own )

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

    function moveFund(address payable _to, uint _amount) onlyowner  public {
        uint balance = address(this).balance;
        uint amount = _amount;
```

(continues on next page)

(continued from previous page)

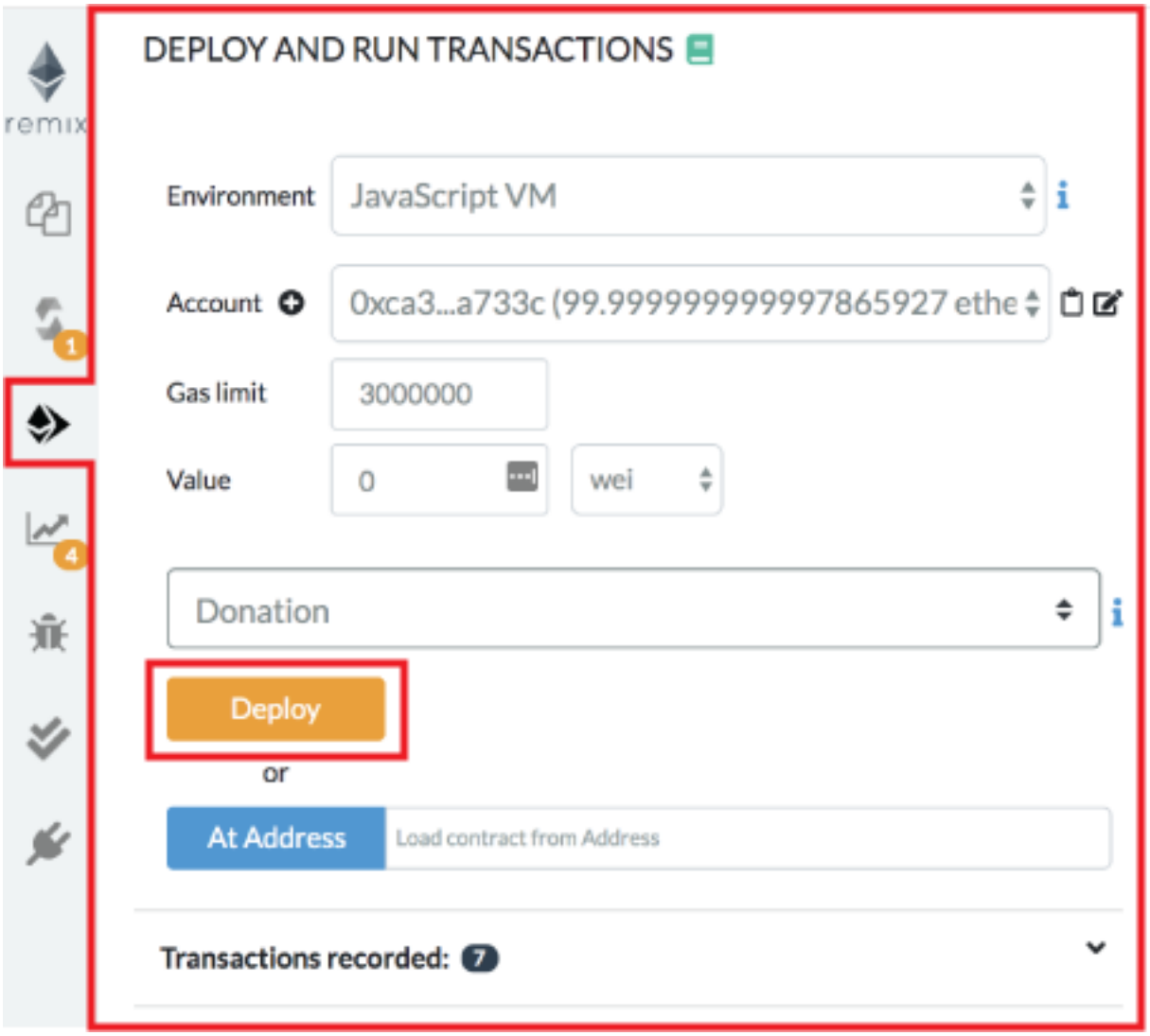
```
        if (_amount <= balance) {
            if (_to.send(balance)) {
                emit fundMoved(_to, _amount);
            } else {
               revert();
            }
        } else {
            revert();
        }
    }

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

- Make a new file in Remix and copy the code above into it.
- Compile the code.
- Go to the Run & Deploy module.

For the purpose of this tutorial, we will run the `JavaScript VM`.

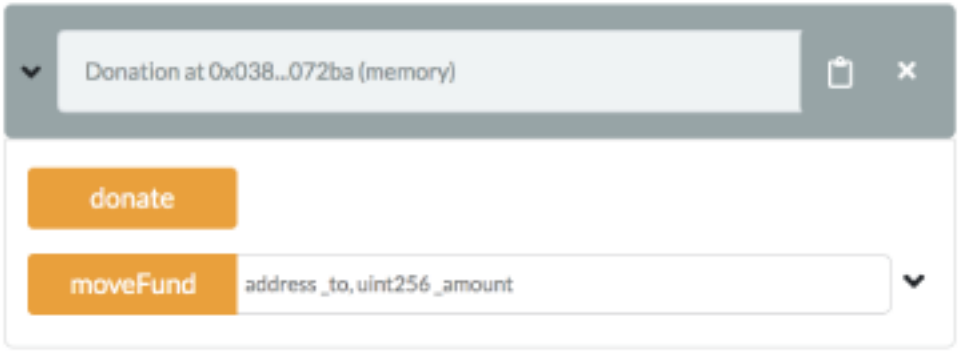
- Deploy the contract:

Click the `Deploy` button



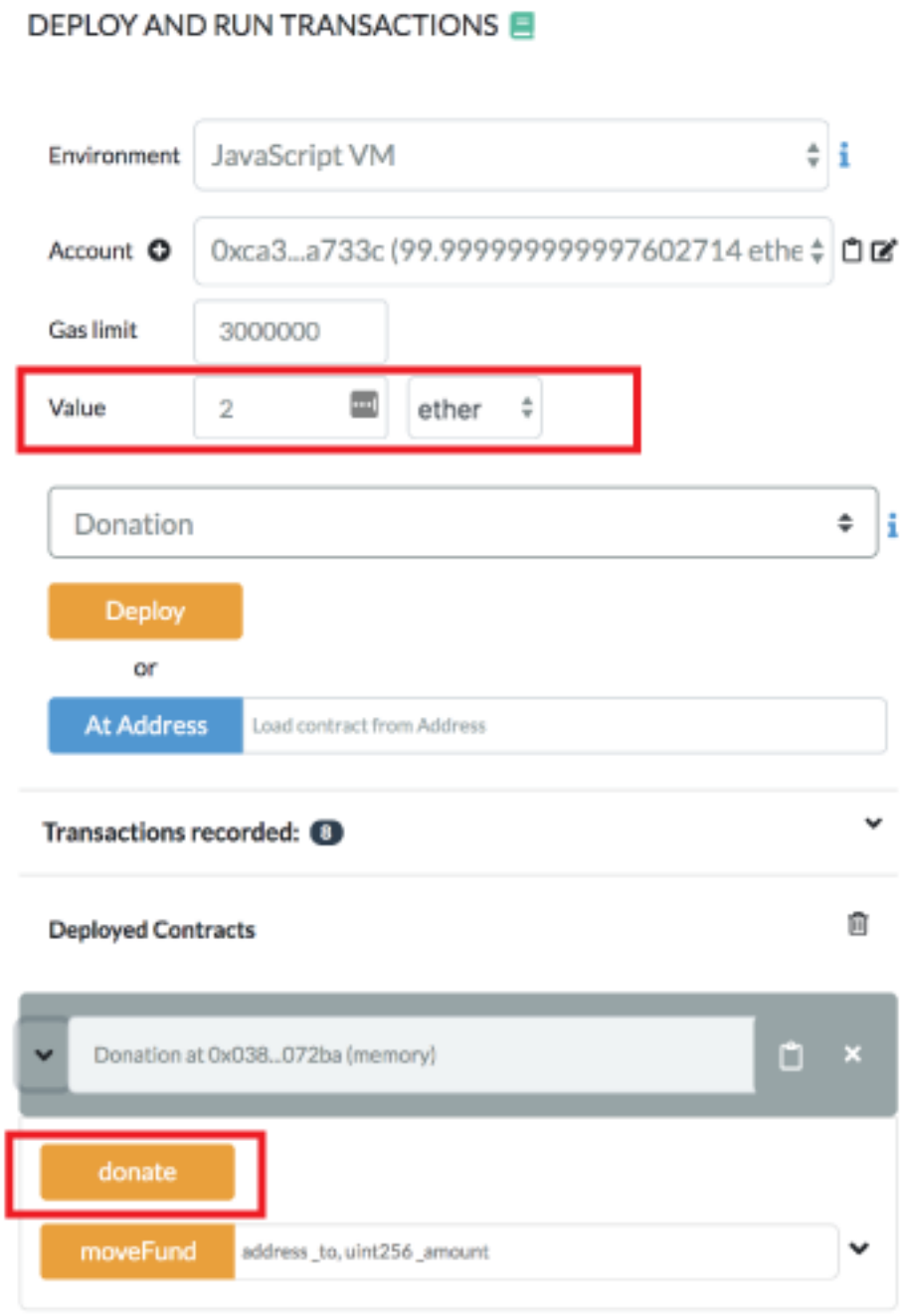

You'll see the deployed instance (AKA the udapp).


Then open it up (by clicking the caret).




We are going to call the `Donate` function and will send 2 Ethers.

To do this: in the value input box put in **2** and **select Ether** as the unit (DO NOT LEAVE THE DEFAULT unit as **gwei** or the change will be hard to detect).




Then click the `Donate` button.

This will send the Ether to the 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**.


But before we get to the actual debugging tool, the next section shows how to start a debugging session directly from the Debugger.

## 21.2 Initiate Debugging from the Debugger

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

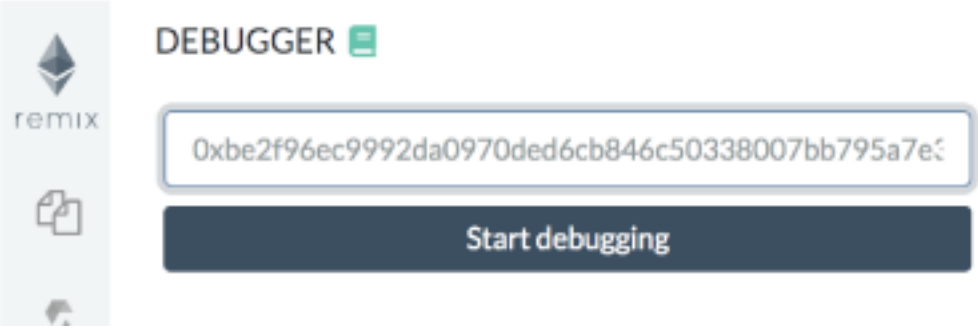
If you don't see the bug icon, go to the plugin manager and activate the debugger.

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

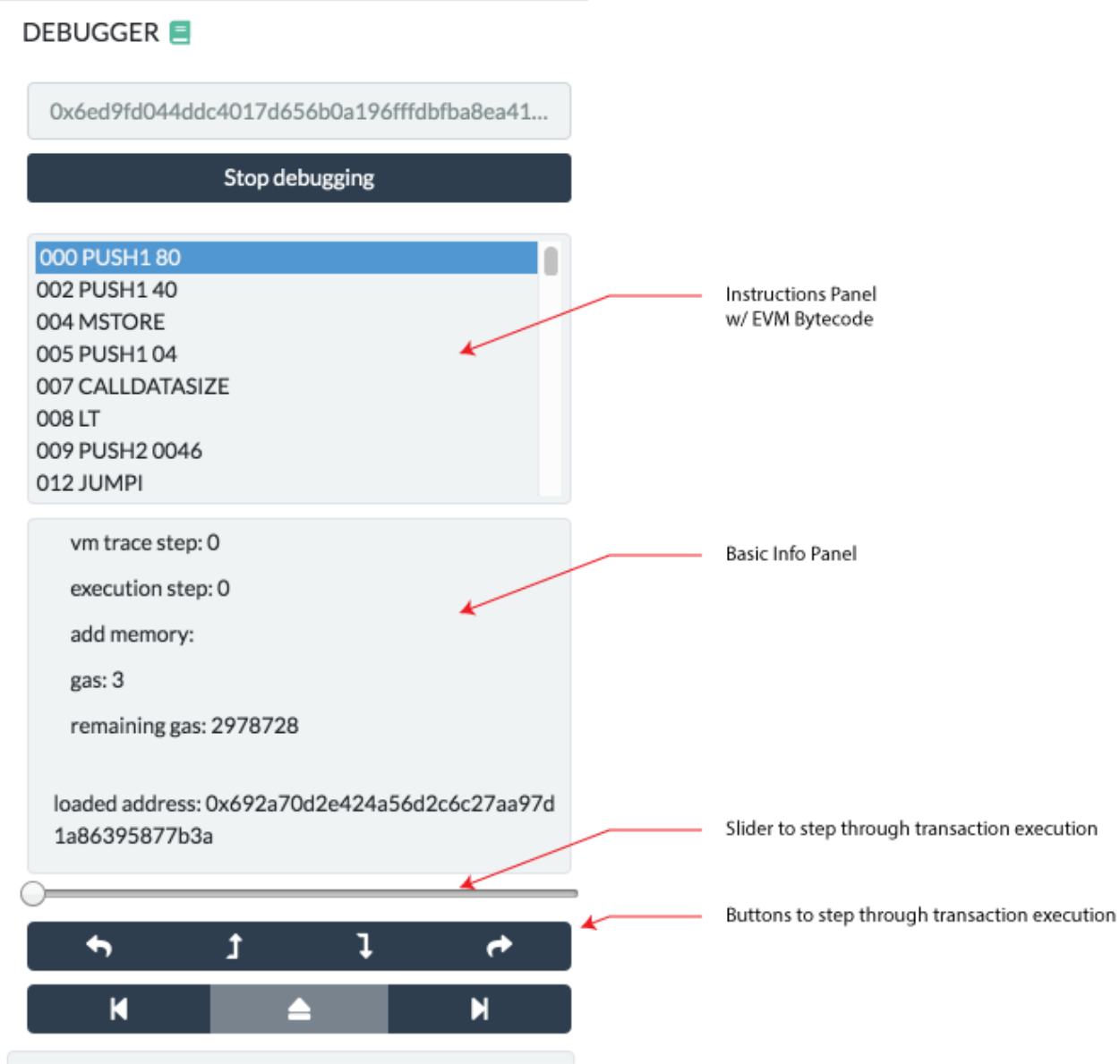
To find a transaction hash:

1. Go to a transaction in the terminal.
2. Click a line with a transaction - to expand 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.




## 21.3 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.

### 21.3.1 More explaination of what these buttons do.

1. Step Into
2. Step Over Into

11 panels give detailed information about the execution:

### 21.3.2 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 coarser 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.

### 21.3.3 Solidity Locals

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

### 21.3.4 Solidity State

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

### 21.3.5 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 & it displays the all the storage changes)

### 21.3.6 Reverted Transaction

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

### 21.3.7 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 a 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 a second time for assigning the actual value.

Here's an example of this issue. If 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 breakpoints are set for the lines

`uint p = 45;`

`m = 89;`

`uint l = 34;`

then clicking on the `Jump to the next breakpoint` button 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 22

# Importing & Loading Source Files in Solidity

There are two main reasons for loading external files into Remix:

- **to import a library or dependency** (for files you will NOT be editing)
- **to load some files for manipulation, editing and play** (for files you might want to edit)

## 22.1 Importing a library or dependency

When importing from NPM, or a URL (like github, a IPFS gateway, or a Swarm gateway) you do not need to do anything more than use the `import` statement in your contract. The dependencies do not need to be "preloaded" into the File Explorer's current Workspace before the contract is compiled.

Files loaded from the import statement are placed in the **Files Explorer's** current Workspace's `.deps` folder.

Under the hood, Remix checks to see if the files are already loaded in the **.deps** directory. If not, it gets them via unpkg if it is an NPM lib.

Here are some example import statements:

### 22.1.1 Import from NPM

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
```

```
import "@openzeppelin/contracts@4.2.0/token/ERC20/ERC20.sol";
```

**Note:** In the example above, **@openzeppelin** is the name of the npm library. In the following example the library's name does not begin with an @ - but Remix will go and check npm for a library of that name.

```
import "solidity-linked-list/contracts/StructuredLinkedList.sol";
```

### 22.1.2 Import from a Github URL

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

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 above imports from OpenZeppelin Contracts v2.5.0.

### 22.1.3 Import from Swarm

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

### 22.1.4 Import from IPFS

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

### 22.1.5 Importing a local file not in .deps

To import a file NOT in the **.deps** folder, use a relative path (**./**). For example:

```
import "./myLovelyLovelyLib.sol";
```

**Note:** It is not possible to import across Workspaces.

### 22.1.6 Importing a file from your computer's filesystem

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

### 22.1.7 More about the import keyword

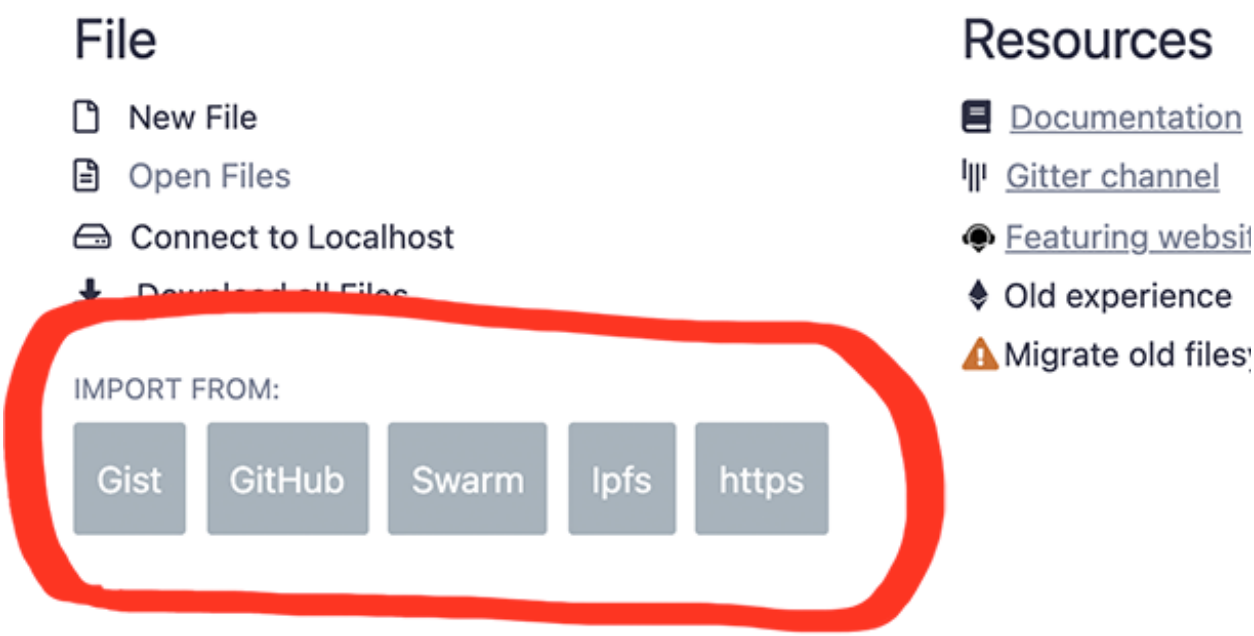
For a detailed explanation of the `import` keyword see the Solidity documentation

## 22.2 Importing a files for manipulation

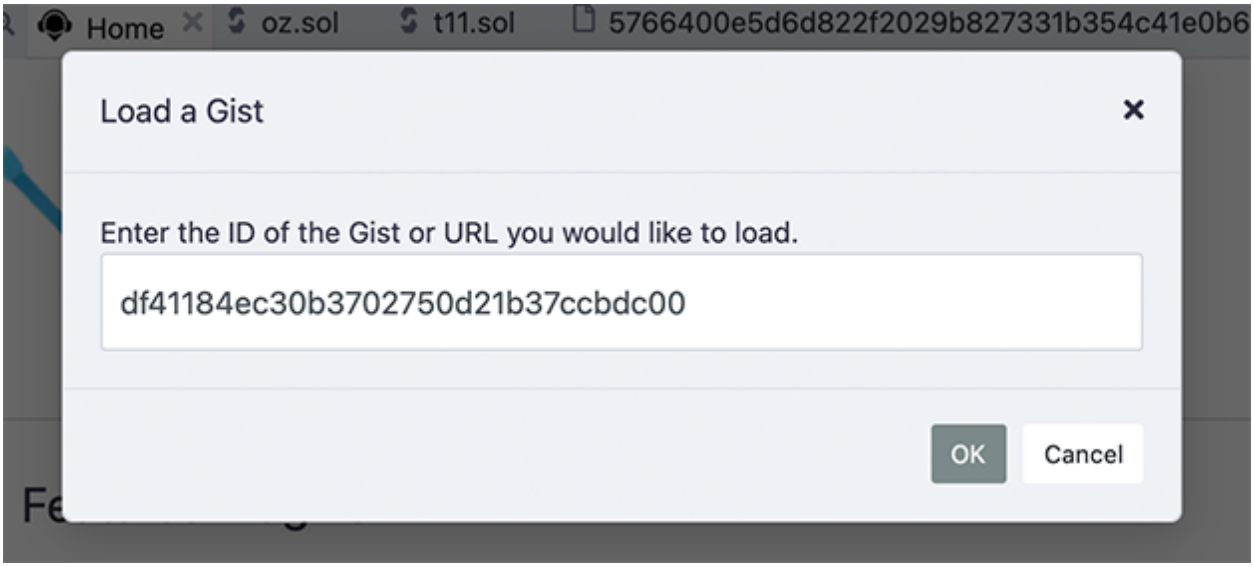
When importing from the home tab widgets or with a remix command in the console, the files are placed in the **root of the current Workspace** inside a folder the shows their source - eg github or gists.

### 22.2.1 Import buttons on the Remix home tab

The Gist, Github, Swarm, IPFS, & HTTPS buttons are to assist in getting files into Remix so you can explore.




Clicking on any of the Import buttons will bring up a modal like this one:




No need to wrap the input in quotes.

### 22.2.2 Loading with a remix command in the console

The 2 remix commands for loading are:

- remix.loadurl(url)
- remix.loadgist(id)

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

```
remix.loadgist('5362adf2000afa4cbdd2c6d239e06bf5')
```

### 22.2.3 Accessing files loaded from the Home tab or from a remix command

When you load from github, a folder named `github` folder is created in the root of your current workspace. To import a file from the `github` folder, you would use a command like this:

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

Notice that this import statement doesn't include the version information that was in the remix.load(url) command. So it is recommended that you use the methods described at the top of this page for importing dependencies that you are not intending to edit.

Assume the .sol file that contained the import statement above is in the contracts folder. Notice that this import statement didn't need to traverse back to the github folder with a relative path like: **../github**.

CHAPTER 23

# Remix Commands

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

**remix**: Remix has a number of CLI commands for loading & executing file in a workspace. See the list below.

**ethers**: Remix IDE enables the use of ethersjs commands. See the Ethers docs for the full list.

**web3**: Remix IDE enable the use of web3js commands. See the Web3js docs for the full list.

**swarmgw**: This library can be used to upload/download files to Swarm via https://swarm-gateways.net/.

## 23.1 Remix Commands

**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.

## 23.2 A few Ethers JS examples

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

**ethers.utils**: The utility functions exposed in both the ethers umbrella package and the ethers-utils. eg ethers.utils.formatBytes32String( text )

## 23.3 A few Web3 JS examples

**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.providers**: Contains the current available providers.

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

## 23.4 A few Swarm examples examples (these will be updated soon)

**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/

CHAPTER 24

# Running JS Scripts in Remix

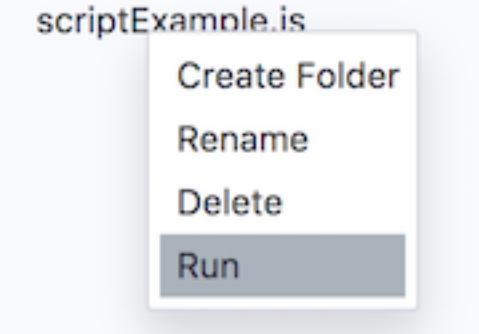
Remix accepts async/await scripts to run web3.js or ethers.js commands. The script needs to be wrapped in a self executing function.

## 24.1 Why run JavaScript Scripts in Remix?

- To mimic how the front-end of your dapp will use web3.js or ethers.js
- To quickly deploy and interact with a bunch of instances of a contract without going through the Remix GUI.
- To run some tests on a previous deployed contract.

## 24.2 Setup

1. These scripts will need to access the contract's ABI. The ABI is located in the contract's metadata file. Make sure that this metadata file will be created by going to the **Settings** module and checking that the **Generate contract metadata** option is indeed **checked**.
2. Compile a Solidity file - to generate the contract metadata.
3. In the Deploy & Run plugin, choose the Environment.
    - Async/await scripts work on in all of the Environments: the JavascriptVM, Injected Web3, and Web3 Provider.
4. Write the script - see below for an example.
5. To run the script - either **(a)** make the script the **active file in the editor** and run `remix.exeCurrent()` in the console OR **(b)** right click on the script **in the files explorer** to get the popup context menu (see the image below) and select the **run** option.




## 24.3 JS Scripts in the File Explorers

In the **scripts** folder of a **workspace**, there are 2 example files: one using **web3.js** and the other using **ethers.js**.

## 24.4 An Example Script

The example below deploys a solidity contract named **CustomERC20.sol**. This example is using the web3.js library. Ethers.js could also be used.

For more information about this example, please see: running async/await scripts

```
(async () => {
  try {
    console.log('deploy...')

    // Note that the script needs the ABI which is generated from the compilation␣
↪artifact.
    const metadata = JSON.parse(await remix.call('fileManager', 'getFile', 'browser/
↪artifacts/CustomERC20.json'))
    const accounts = await web3.eth.getAccounts()

    let contract = new web3.eth.Contract(metadata.abi)

    contract = contract.deploy({
      data: metadata.data.bytecode.object,
      arguments: ["Mask", "N95"]
    })

    newContractInstance = await contract.send({
      from: accounts[0],
      gas: 1500000,
      gasPrice: '30000000000'
    })
    console.log(newContractInstance.options.address)
  } catch (e) {
    console.log(e.message)
  }
})()
```

For more script examples, please see Frequently Asked Scripts.

# CHAPTER 25

# Frequently Asked Scripts

Deploy with web3.js

```
(async () => {
  try {
    console.log('deploy...')

    // Note that the script needs the ABI which is generated from the compilation artifact.
    const metadata = JSON.parse(await remix.call('fileManager', 'getFile', 'browser/artifacts/CustomERC20.json'))
    const accounts = await web3.eth.getAccounts()

    let contract = new web3.eth.Contract(metadata.abi)

    contract = contract.deploy({
      data: metadata.data.bytecode.object,
      arguments: ["Mask", "N95"]
    })

    newContractInstance = await contract.send({
      from: accounts[0],
      gas: 1500000,
      gasPrice: '30000000000'
    })
    console.log(newContractInstance.options.address)
  } catch (e) {
    console.log(e.message)
  }
})()
```

Deploy with Ethers

```
(async function() {
  try {
```

(continues on next page)

(continued from previous page)

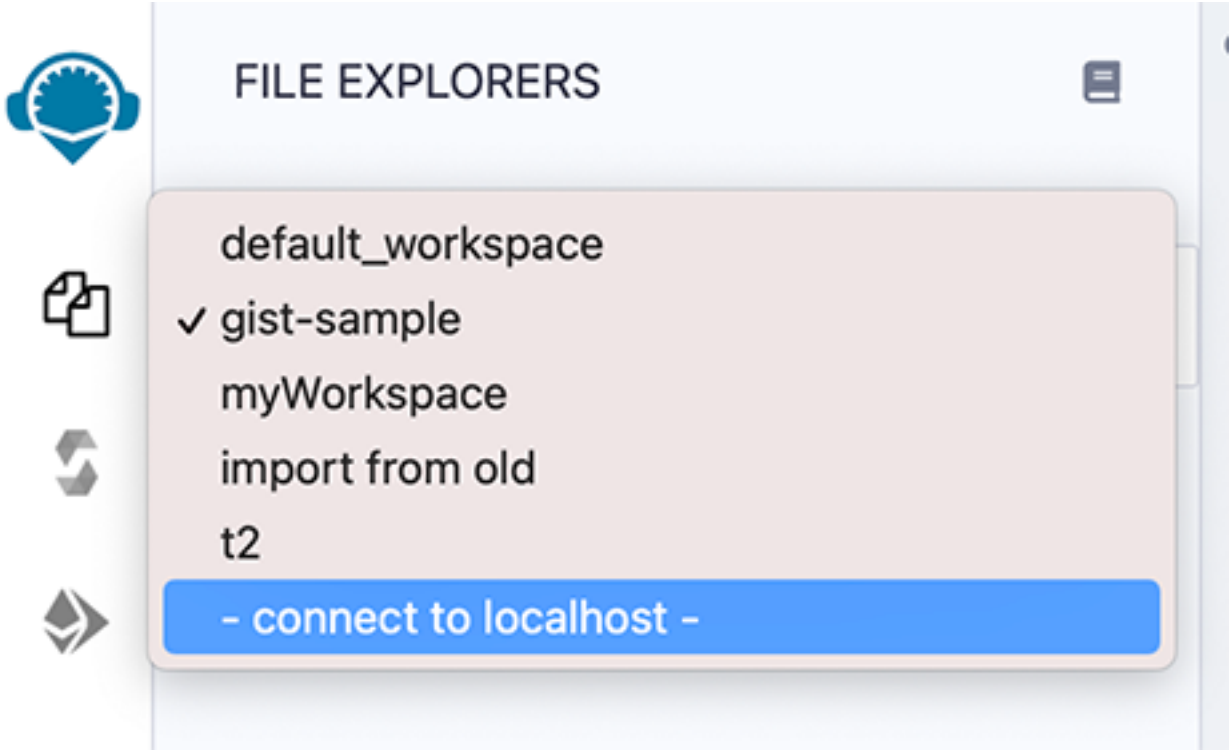
```
    const metadata = JSON.parse(await remix.call('fileManager', 'getFile', 'browser/
↪artifacts/CustomERC20.json'))
    // the variable web3Provider is a remix global variable object
    const signer = (new ethers.providers.Web3Provider(web3Provider)).getSigner()
    // Create an instance of a Contract Factory
    let factory = new ethers.ContractFactory(metadata.abi, metadata.data.bytecode.
↪object, signer);
    // Notice we pass the constructor's parameters here
    let contract = await factory.deploy('Mask', 'N95');
    // The address the Contract WILL have once mined
    console.log(contract.address);
    // The transaction that was sent to the network to deploy the Contract
    console.log(contract.deployTransaction.hash);
    // The contract is NOT deployed yet; we must wait until it is mined
    await contract.deployed()
    // Done! The contract is deployed.
    console.log('contract deployed')
  } catch (e) {
    console.log(e.message)
  }
})();
```

CHAPTER 26

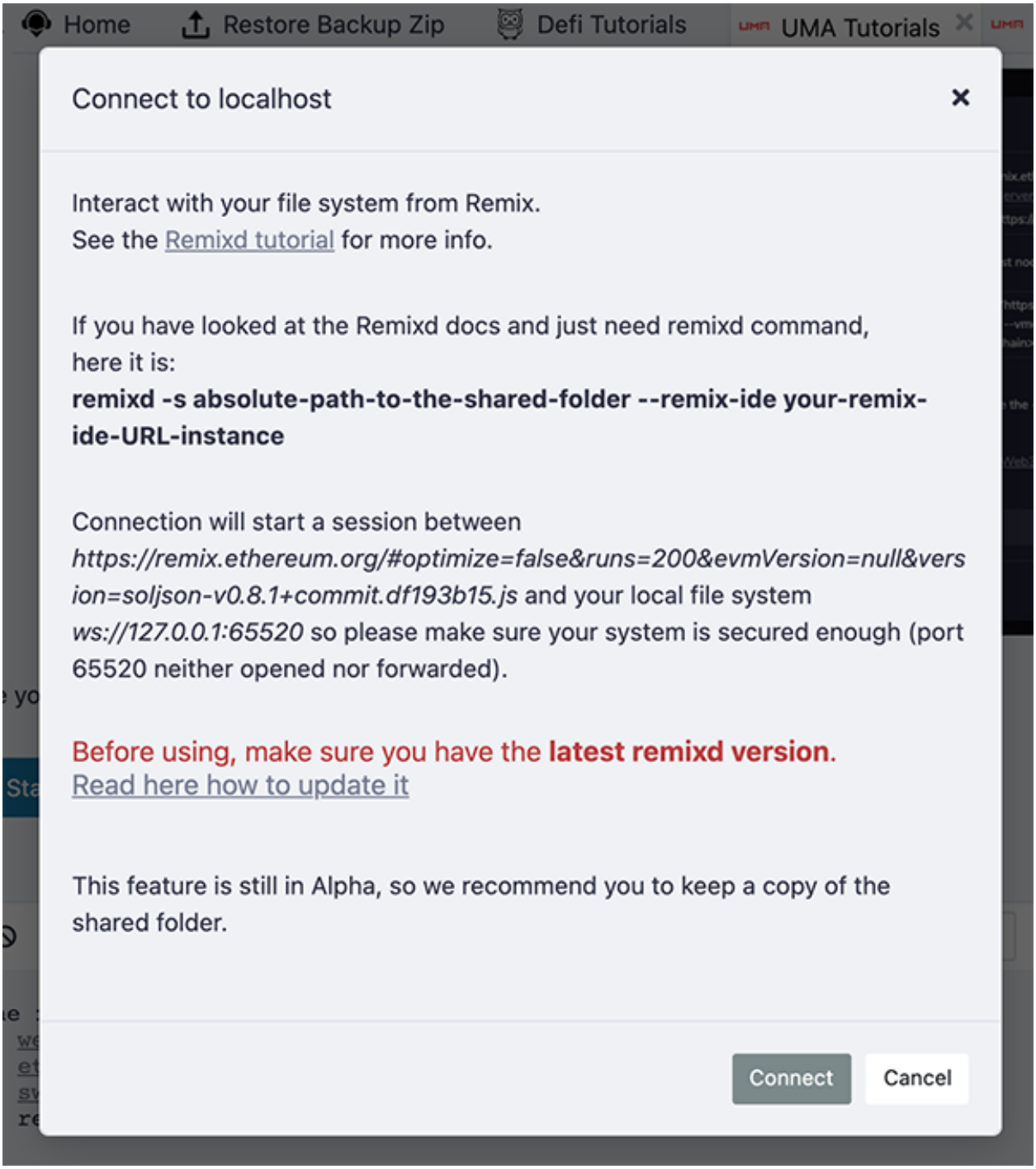
# Remixd: Access your Local Filesystem

To give the Remix IDE (the web app) access to a folder on your computer, you need to use **Remixd** - the plugin along with **remixd** - the cli/npm module.

The **Remixd** plugin can be activated from the plugin manager or in the **File Explorers** - see the image below. The **connect to local host** - will activate the **Remixd** plugin.




Once you click **connect to local host** or activate Remixd from the **Plugin Manager**, a modal will come up:




The Remixd plugin is a **websocket plugin** and it has no UI other than this modal dialog box - so you won't see a Remixd icon in the icon panel.

Before you hit **Connect**, you need to install the **remixd** NPM module and run the **remixd** command.

The code of `remixd` is here .

## 26.1 remixd Installation

**remixd** can be globally installed using the following command: `npm install -g @remix-project/remixd`

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

**NOTE:** The npm address as well as the github repo of remixd have changed - in both cases moving under **remix-project**.

- **Github** address is: **https://github.com/ethereum/remix-project/tree/master/libs/remixd**
- **NPM** address is: **https://www.npmjs.com/package/@remix-project/remixd**

## 26.2 Find your version of remixd

The command: `remixd -v` or `remixd --version` will return your version number.

**If this command does not work, then you have an outdated version!**

## 26.3 Update to the latest remixd

Because **remixd** creates a bridge from the browser to your local filesystem, it is important that you have the latest version of script.

For users who had installed the version of remixd from the old NPM address or for users who do not know which NPM address they had installed it from, run these 2 steps:

1. uninstall the old one: **npm uninstall -g remixd**
2. install the new: **npm install -g @remix-project/remixd**

For users who know that they have a remixd version installed from @remix-project/remixd then just run:

**npm install -g @remix-project/remixd**

## 26.4 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.

### 26.4.1 HTTP vs HTTPS in the remixd command

If your browser is on https://remix.ethereum.org (**secure http**) then use https in the command: `remixd -s <absolute-path-to-the-shared-folder> --remix-ide https://remix.ethereum.org`

Or if you are using **http** in the browser, then use **http** in the remixd command.

### 26.4.2 Read/Write permission & Read-only mode

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.

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

## 26.5 Ports Usage

`remixd` functions by making websocket connections with Remix IDE on different ports. Ports are defined according to specific purpose. Port usage details are as:

- **65520** : For `remixd` websocket listener, to share local file system with Remix IDE. Shared folder will be loaded in the Remix IDE `File Explorers` workspace named `localhost`
- **65523** : For `slither` websocket listener, to enable the Slither Analysis using Remix IDE `Solidity Static Analysis` plugin
- **65522** : For `hardhat` websocket listener, to enable the Hardhat Compilation using Remix IDE `Solidity Compiler` plugin, if shared folder is a Hardhat project.

**Note:** Please make sure your system is secured enough and these ports are not opened nor forwarded.

## 26.6 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.
- To minimize the risk, Remixd can **ONLY** bridge between your filesystem and the Remix IDE URLS - including:

```
https://remix.ethereum.org
https://remix-alpha.ethereum.org
https://remix-beta.ethereum.org
http://remix.ethereum.org
http://remix-alpha.ethereum.org
http://remix-beta.ethereum.org
package://a7df6d3c223593f3550b35e90d7b0b1f.mod
package://6fd22d6fe5549ad4c4d8fd3ca0b7816b.mod
https://ipfsgw.komputing.org
```

(the package:// urls in the list above are for remix desktop)

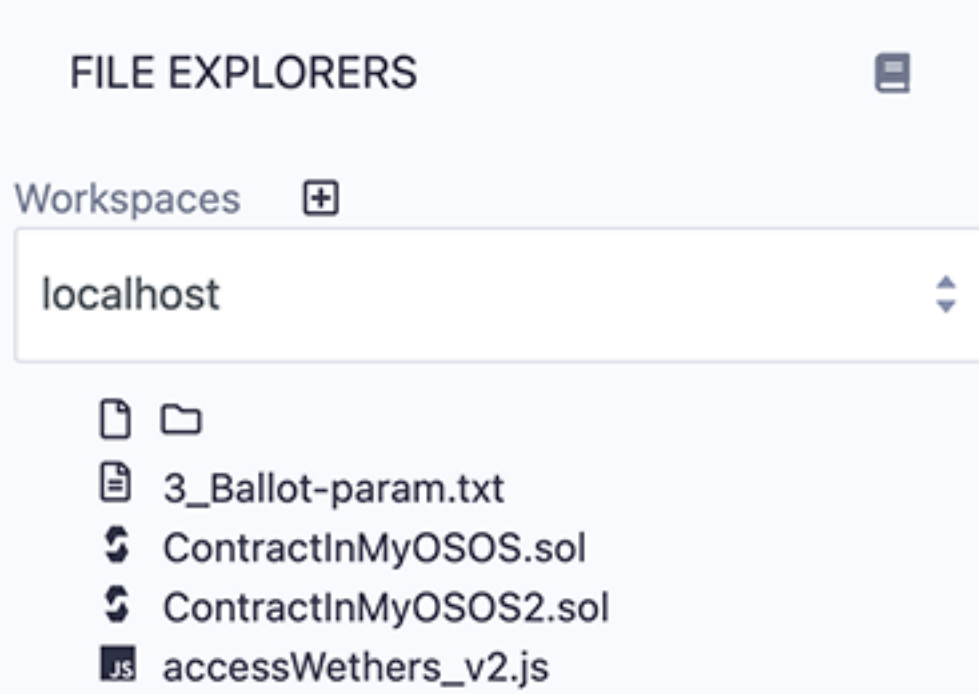
## 26.7 Clicking Connect on the modal.

Clicking on the **Connect** button on the Remixd modal (see the image above), will attempt to start a session where your browser can access the specified folder on your computer's filesystem.

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 visible in the File Explorer's workspace under **localhost**.




## 26.8 Creating & deleting folders & files

Clicking on the **new folder** or **new file** icon under **localhost** will create a new file or folder in the shared folder. Similarly, if you **right click** on a file or folder you can **rename** or **delete** the file.

## 26.9 Closing a remixd session

In the terminal where **remixd** is running, typing `ctrl-c` will close the session. Remix IDE will then put up a modal saying that **remixd** has stopped running.

CHAPTER 27

# FAQ

## 27.1 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.

## 27.2 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.

## 27.3 General

**Q:** Where do plugin developers go with their questions?

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

## 27.4 Analytics

**Q:** What information does Remix save when Matomo Analytics is enabled?

**A:** We want to know:

- Which plugins get activated & deactivated
- If users check the box to publish a contract's metadata when deploying
- Which themes are used/used most/not used at all
- The usage of the links to documentation
- On the homepage, which file importing buttons are used

**Q:** Is it opt-in or opt-out?

**A:** We use Matomo as an opt-in analytics platform.

**Q:** Where is the info stored? Is the info shared with 3rd parties?

**A:** All data collected through Matomo is stored on our own server.No data is given to third parties.

We respect your privacy and do not collect nor store any personally identifiable information (PII).

**Q:** What does Remix do with this info?

**A:** Our goal is to understand how many users we have, what plugins people are using, what is not getting used, what is not being used to its full potential.

With this understanding, we can make adjustments to the UI as well as providing more tips and documentation. It's a way of getting constant anonymous feedback from our users.

**Q:** After I agree opt-in, can I change my mind?

**A:** You can turn off or on Matomo in the Settings panel. There are no consequences for not opting-in or opting-out.

CHAPTER 28

# Remix URLs & Links with Parameters

## 28.1 Main Remix URLs

- Remix IDE Online is located at https://remix.ethereum.org.
- The alpha version of remix is located 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.
- Remix Desktop is an Electron App. Here is the release page.
- Remix has a VSCode extension called Ethereum Remix.
- The Remix twitter account is EthereumRemix.
- The Remix Project Medium publication is: https://medium.com/remix-ide.
- The Remix Project website introduces the different facets of our project.
- The Remix Gitter Channel is a forum to post your questions about Remix.

## 28.2 Customize Remix with URL Parameters

There are many ways to customize Remix IDE by using url parameters. Here are some options:

- Activate or deactivate a **list of plugins to be activated** - and specify which plugin gains the "focus". *SEE MORE*
- Send **commands to a plugin** - once the plugin loads. *SEE MORE*
- *Load a GIST*, *a file via a url* or a *base64 encoded string* into Remix's Editor.
- Specify **the theme** (Dark or Light). *SEE MORE*
- Specify which panels should be **minimized** - useful when embedding Remix in your site. *SEE MORE*
- Select the **version of the Solidity** compiler, enable/disable the **optimizer**, turn on auto compile or choose the language for the Solidity compiler. *SEE MORE*

### 28.2.1 Activating a list of plugins

The following example contains the url parameter **activate** followed by **a comma separated list of plugins**.

The last plugin in the list will gain the focus.

When you use the activate list, all other plugins that a user had loaded will be deactivated. This does not apply to the file explorer, the plugin manager, and the settings modules because these are never deactivated.

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

Note: a plugin is called by its **name** as specified in its profile. There are 3 types of plugins:

1. **Native Mandatory Plugins** that are always loaded (so you don't need to activate them using the url parameter **activate**). These include: **fileManager**, **settings**, **manager** (the plugin manager), and **udapp** (deploy & run).
2. **Native Optional Plugins** that are loaded on demand: **debugger**, **hardhat-provider**, **solidity**, **solidityStaticAnalysis**, **solidityUnitTesting**, and **vyper**
3. **External Plugins** to get these plugins' names, please go to https://github.com/ethereum/remix-plugins-directory/tree/master/plugins.

### 28.2.2 Deactivating a list of plugins

```
https://remix.ethereum.org/?#deactivate=debugger
```

### 28.2.3 Minimizing Remix panels

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

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

To minimize just the side panel, use this URL:

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

To minimize just the terminal, use this URL:

```
https://remix.ethereum.org/?#minimizeterminal=true
```

### 28.2.4 Specifying a theme

To link to Remix with a theme specified use this url:

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

### 28.2.5 A URL example combining multiple parameters

To link to Remix with the a list of plugins activated and with:

- the Learneth gaining the side panel's focus (because it is the last in the list)
- the Light theme loaded

- the terminal minimized
- optimize off

use this url:

```
https://remix.ethereum.org/?#activate=solidity,solidityUnitTesting,LearnEth&
↪theme=Light&minimizeterminal=true&optimize=false&evmVersion=null&version=soljson-v0.
↪6.6+commit.6c089d02.js
```

## 28.3 Pass commands to a plugin's API via a url param

The URL parameter to issue a command is `call`. Following the `call` is a // (double slash) separated list of arguments.

```
call=plugin_name//function//parameter1//paremeter2
```

### 28.3.1 An example using call

The URL below uses `activate` & `call`. It **activates** a number of plugins and **calls** the File Explorers to tell it to load one of the default Remix files:

```
https://remix.ethereum.org/?#activate=defiexplorer,solidity&call=fileManager//open//
↪contracts/3_Ballot.sol
```

### 28.3.2 Load a specific tutorial in the LearnEth plugin:

```
https://remix.ethereum.org/?#activate=solidityUnitTesting,solidity,LearnEth&
↪call=LearnEth//startTutorial//ethereum/remix-workshops//master//proxycontract
```

## 28.4 Load a file via a URL into the Editor

The `url` parameter takes a URL, loads it into the Editor and saves it into the code-sample workspace of the File Explorer:

```
https://remix.ethereum.org/#url=https://github.com/ethereum/remix-project/blob/master/
↪apps/remix-ide/contracts/app/solidity/mode.sol
```

## 28.5 Load an encoded base64 string into a .sol file in the Editor

The `code` parameter takes an encoded base64 string and loads it into the Editor as a .sol file and saves to the code-sample workspace of the File Explorer:

```
https://remix.ethereum.org/?
↪#code=Ly8gU1BEWC1MaWNlbnNlLUlkZW50aWZpZXI6IE1JVAoKcHJhZ21hIHNvbGlkaXR5IDAuOC40OwoKLyoqCiAqIEB0aXRs
```

## 28.6 Load a GIST

The URL parameter here is `gist`.

```
https://remix.ethereum.org/?gist=0fe90e825327ef313c88aedfe66ec142
```

### 28.6.1 Load a GIST and have it be visible in the Editor:

Using both `gist` & `call`

```
https://remix.ethereum.org/?#activate=solidity,debugger&
↪gist=0fe90e825327ef313c88aedfe66ec142&call=fileManager//open//browser/gists/
↪0fe90e825327ef313c88aedfe66ec142/gridMix4.sol
```

### 28.6.2 Load a GIST, have it be visible in the Editor & load a list of plugins:

```
https://remix.ethereum.org/?#activate=solidity,LearnEth&
↪gist=0fe90e825327ef313c88aedfe66ec142&call=fileManager//open//browser/gists/
↪0fe90e825327ef313c88aedfe66ec142/gridMix4.sol
```

## 28.7 Load a specific version of the Solidity compiler:

```
https://remix.ethereum.org/?#version=soljson-v0.6.6+commit.6c089d02
```

**Note:** you need to specify both the Solidity version and the commit.

## 28.8 Load a custom Solidity compiler:

```
https://remix.ethereum.org/#version=https://solidity-blog.s3.eu-central-1.amazonaws.
↪com/data/08preview/soljson.js
```

## 28.9 Turn on autoCompile:

```
https://remix.ethereum.org/#autoCompile=true
```

## 28.10 Select the language for the Solidity Compiler

Choose YUL or Solidity with the language parameter.

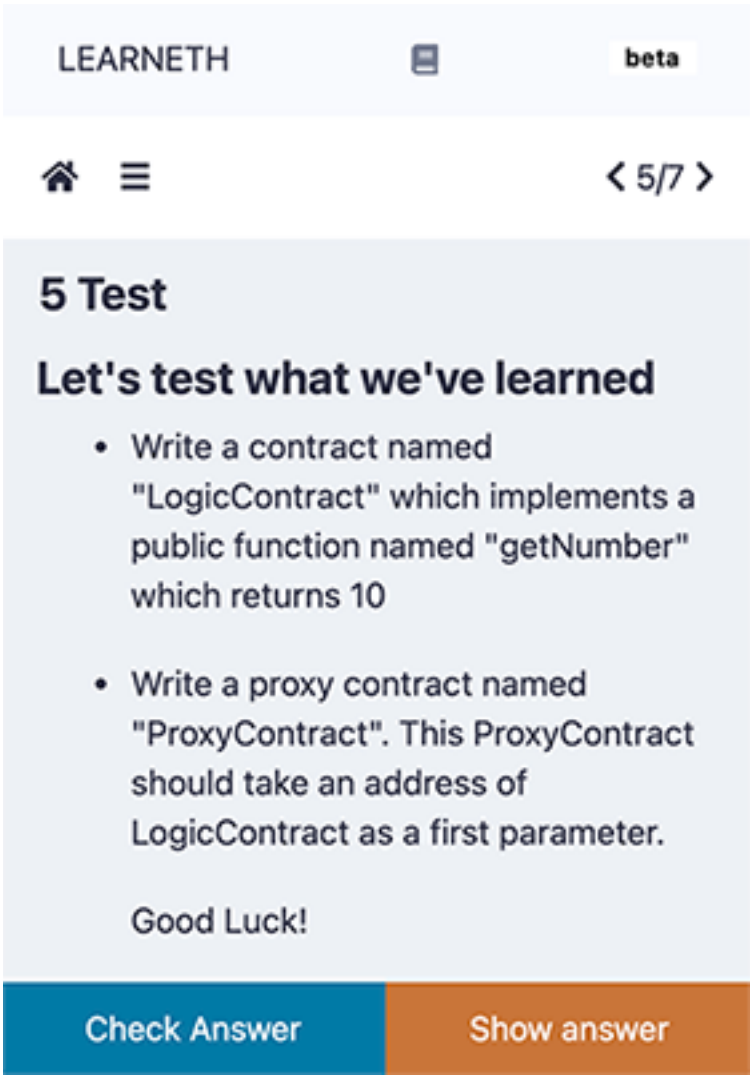
```
https://remix.ethereum.org/#language=Yul
```

CHAPTER 29

# Remix Tutorials with Learneth

**Learneth** is a tutorial platform integrated into Remix.

Tutorials can contain quizzes for testing students' work. These quizzes are run by Solidity Unit Tests.




We have a growing set of tutorials on our repo- but anyone can build tutorials on their own repos and have their students load them up!

The tutorials contain .md files for instructions and can also contain example files, Solidity Unit Test files for quizzes, as well as answer files for a quizzes.

## 29.1 Opening Learneth & associated links

Learneth is a plugin - so to access it, you need to activate the Learneth plugin in the Plugin Manager. Alternatively - this link will active it: click this link.

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

This link will activate Learneth and then will open a specific tutorial - in this case it will load the **proxy contract** tutorial:

```
https://remix.ethereum.org/?#activate=udapp,solidity,LearnEth&call=LearnEth//
↪startTutorial//ethereum/remix-workshops//master//proxycontract
```

**NOTE:** For other tricks about Remix URLs with parameters, go here: locations.

## 29.2 Learneth Tutorials

Here is the current list of Learneth Tutorials

***Beginner***

```
Remix Basics
Intro to Solidity
```

***Intermediate***

```
Basic Use of web3.js
The Recorder
```

***Advanced***

```
All About Proxy Contracts
Deploy with Libraries
Opcodes in the Debugger
```

## 29.3 Learneth & Tutorial Repos

The code for the Learneth plugin is located here: https://github.com/bunsenstraat/remix-learneth-plugin/blob/master/docs/index.rst

Documentation for creating your own tutorials is located here: https://remix-learneth-plugin.readthedocs.io/en/latest/index.html

Remix maintains and curates this repo of Learneth tutorials: https://github.com/ethereum/remix-workshops

CHAPTER 30

# 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 31

# 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.