Remix Documentation

Release 0.1

yann300

Aug 16, 2019

Quick start

VIDE (Beta version) is developed based on Remix. VIDE is a powerful, open source tool that helps you write Solidity contracts straight from the browser. Written in JavaScript, VIDE supports usage in the browser.

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

Our VIDE project with all its features is available at vechainstore.com/ide 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 VIDE and some tutorials to help you get started.

Useful links:

- Solidity documentation
- VeChain GitHub repository
- VeChain community channel (Reddit)
- VeChain community channel (Twitter)
- VeChain community channel (Medium)
- VIDE support channel (Discord)

CHAPTER 1

Packages

This part focuses on using `VIDE`, which is a browser based smart contract IDE. We will basically answer the question: Where can I use / download `VIDE`, and what is the difference between packages?

- An online version is available at https://vechainstore.com/ide. This version is stable and is updated at almost every release.
- Github repository: https://github.com/mobileteamdev/vide-web . The source code is packaged at every release but still need to be built using `npm run build`.

CHAPTER 2

Solidity Editor

The VIDE 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.

image

Here's the list of some important features:

- It display opened files as tabs.
- Compilation Warning and Error are displayed in the gutter
- VIDE 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 3

Compiling contracts

By default VIDE triggers a compilation each time the current file is changed or another file is selected. If the contract has a lot of dependencies and takes a long time to compile, it is possible to disable the autocompilation.

image

After each compilation, a list is updated with all the newly compiled contracts.

Details modal dialog displays detailed information about the current selected contract.

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

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

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

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

CHAPTER 4

Settings

This section displays the current compiler version and allows one to change to another version.

image

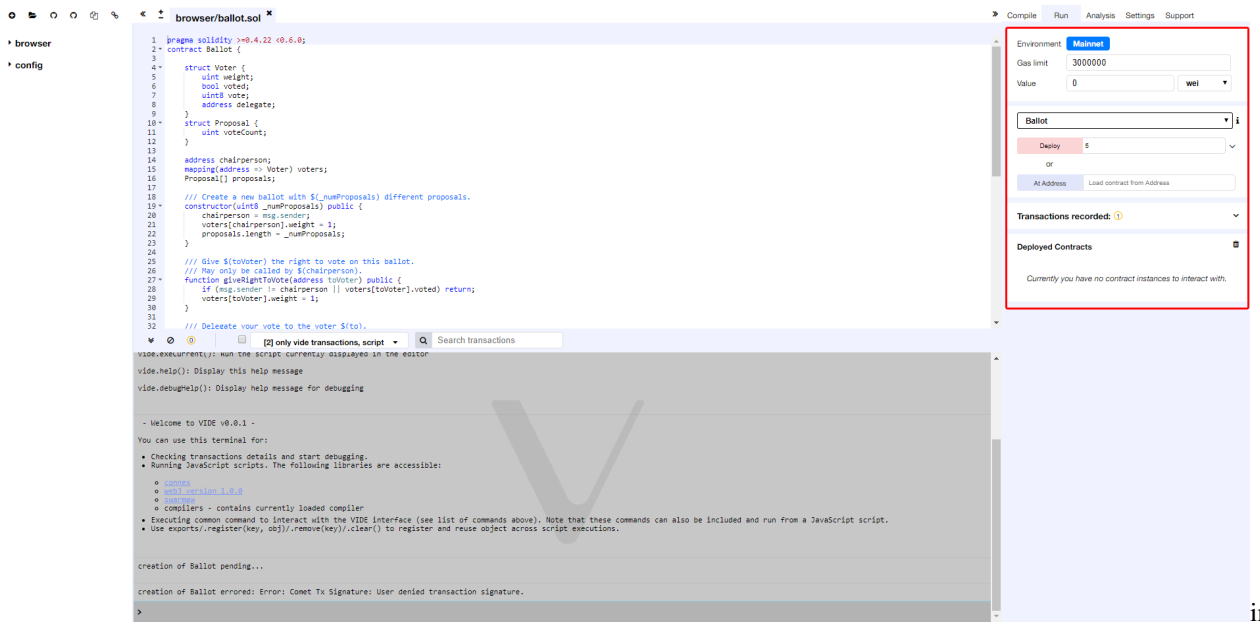
Another important settings:

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

CHAPTER 5

Running transactions

The Run tab is an important section of VIDE. It allows you to send transactions to the current environment.



image

5.1 Run Setup

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

Environment:

- `Mainnet` or `Testnet`: VIDE will connect to an injected Connex provider. `Sync` and `Comet` are example of providers that inject Connex, thus can be used with this option.
- Gas Limit: the maximum amount of gas that can be set for all the transactions created in VIDE.

- Value: the amount of value for the next created transaction (this value is always reset to 0 after each transaction execution).

image

5.2 Initiate Instance

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

- `At Address` assumes the given address is an instance of the selected contract. It is then possible to interact with an already deployed contract. There's no check at this point, so be careful when using this feature, and be sure you trust the contract at that address.
- `Create` send a transaction that deploys the selected contract. When the transaction is mined, the newly created instance will be added (this might take several seconds). Note that if the `constructor` has parameters, you need to specify them.

5.3 Pending Instances

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

5.4 Using the ABI

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

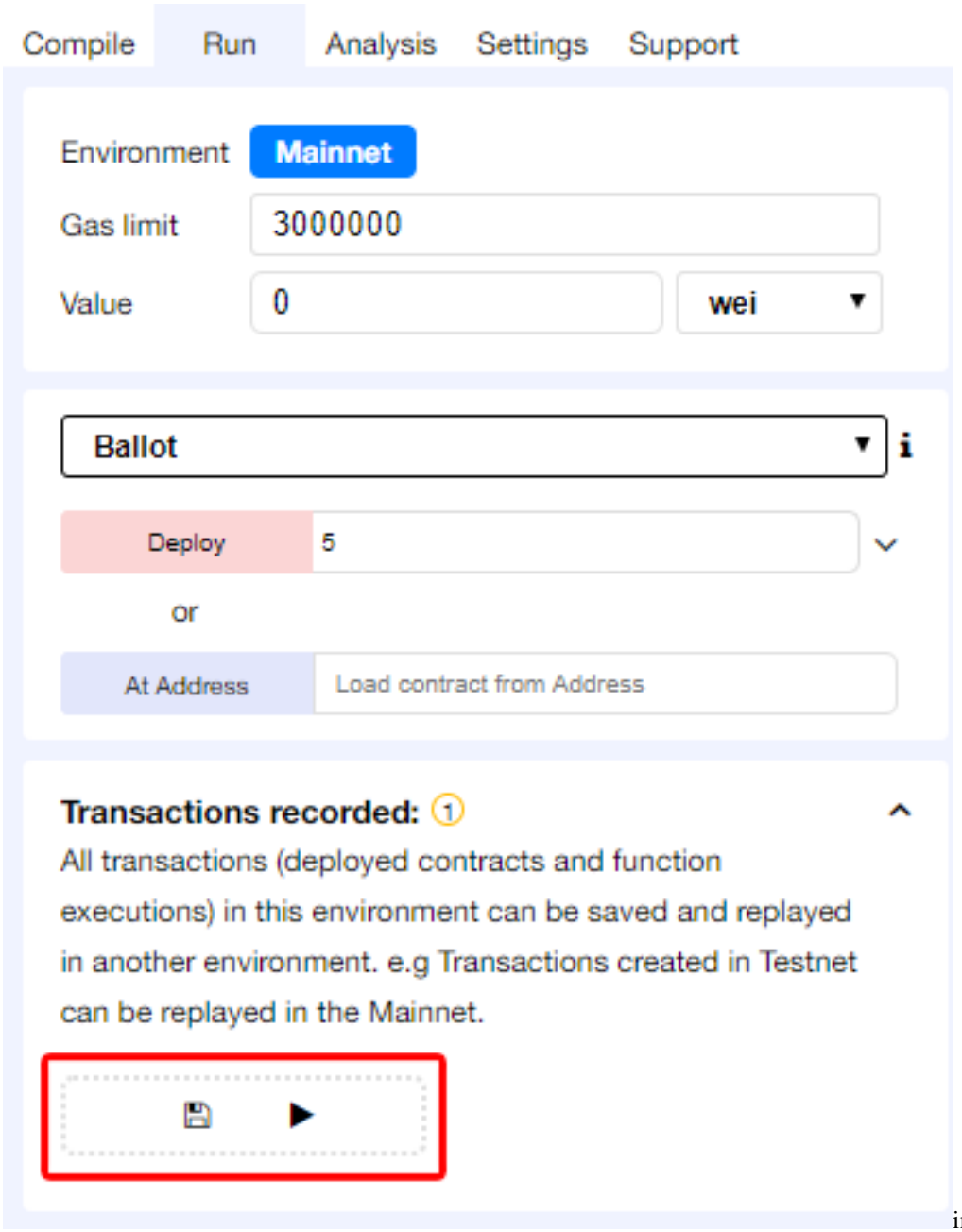
To interact with a contract using the ABI, create a new file in VIDE with extension `*.abi` and copy the ABI content to it. Then in the input next to `At Address`, put the Address of the contract you want to interact with. Click on `At Address`, a new "connection" with the contract will popup below.

5.5 Using the Recorder

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

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

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



image

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

In that specific record, 3 transactions are executed:

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

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

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

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

```
{
"accounts": {
    "account{0}": "0xca35b7d915458ef540ade6068dfe2f44e8fa733c"
},
"linkReferences": {
    "testLib": "created{1512830014773}"
},
"transactions": [
    ...
],
"abis": {
    ...
}
}
```

CHAPTER 6

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).

Several cases apply:

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

For more information see more about Solidity modifier .

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

CHAPTER 7

Build Artifact

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

7.1 Library Deployment

By default VIDE automatically deploy needed libraries.

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

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

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

CHAPTER 8

File Explorer

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

image

Note that clearing the browser storage will permanently delete all the solidity files you wrote.

image

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

8.1 Create new File

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

8.2 Add Local File

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

8.3 Publish to Gist

Publishes all files from the browser folder to a gist. Gist API has changed in 2018 and it unfortunately requires users to be authenticated to be able to publish a gist.

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

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

8.4 Copy to another VIDE instance

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

CHAPTER 9

Debugging

This feature will be available in the next version!

CHAPTER 10

Analysis

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

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

image

Here is the list of analyzers:

Security:

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

- Low level calls: Semantics maybe unclear
- Block.blockhash usage: Semantics maybe unclear

Gas & Economy:

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

Miscellaneous:

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

CHAPTER 11

Terminal

image

Features, available in the terminal:

- It integrates a JavaScript interpreter and the `connex` and `web3` object. It enables the execution of the JavaScript script which interacts with the current context.
- It displays important actions made while interacting with the VIDE (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 VIDE knows (e.g transaction created from the VIDE).
- It allows searching for the data and clearing the logs from the terminal.

CHAPTER 12

Tutorial on debugging transactions with VIDE

This feature will be available in the next version!

CHAPTER 13

Importing Source Files in Solidity

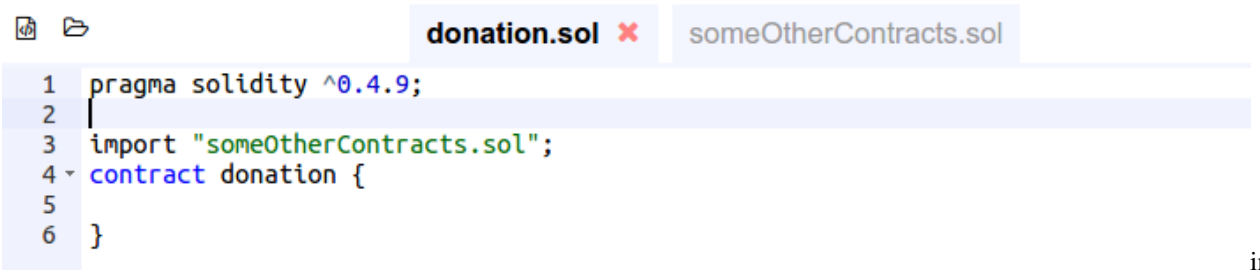
This tutorial will show you how to import local and external files.

The compilation result will also contain contracts implemented in the imported files.

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

13.1 Importing a local file

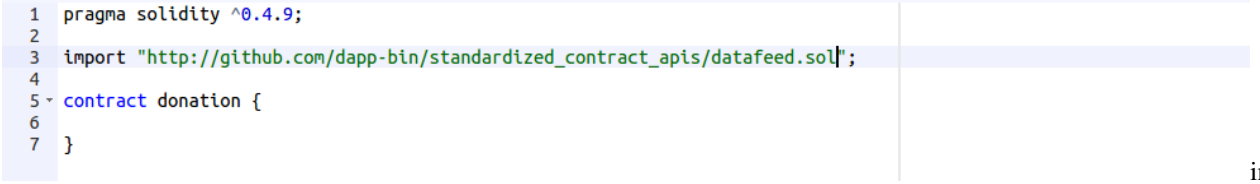
Other files in VIDE can be imported just by specifying their path. Please use ./ for relative paths to increase portability.



image

13.2 Importing from GitHub

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



image

13.3 Importing from Swarm

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

image

CHAPTER 14

Code contribution guide

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

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

To learn more, please visit our GitHub page.

CHAPTER 15

Support tab in VIDE

Have a question, found a bug or want to propose a feature? Please join the discord group Vechain IDE