solitude Documentation

Release 0.0.2

incerto-crypto

Jun 25, 2019

Quickstart

Solitude is a framework to deploy, interact, test and debug your Solidity contracts.

CHAPTER 1

Getting Started

CHAPTER 2

Examples

CHAPTER 3

SolitudeConfiguration

type	*object*		
properties			
• **Project.Name**	Your project's name		
	type	*string*	
	default	MyProject	
• **Project.SourceDir**	Directories where the contract source files are located		
	type	*string*	
	default	./contracts	
• **Project.ObjectDir**	Directory where the (compiled) contract objects are located		
	type	*string*	
	default	./build/contracts	
• **Tools.Directory**	Path to the directory where the downloaded tools will be installed		
	type	*string*	
	default	~/.solitude-dev	
• **Tools.Solc.Version**	solc (Compiler) required version		
	type	*string*	
	default	0.5.2	
• **Tools.GanacheCli.Version**	ganache-cli (Server) required version		
	type	*string*	
	default	6.4.1	
• **Tools.EthLint.Version**	ethlint (Linter) required version		
	type	*string*	
	default	1.2.4	
• **Tools.Required**	List of tools required by your project		
	type	*array*	
	default	['Solc', 'GanacheCli']	
	items		
	•	type	*string*
		enum	Solc, GanacheCli, EthLint
• **Server.Host**	Host on which to start the server		

Continued on next page

Table 1 – continued from previous page

	type	*string*		
	default	127.0.0.1		
• **Server.Port**	Port on which the server is started			
	type	*number*		
	default	8545		
• **Server.Accounts**	Initial accounts and balances for the server			
	type	*array*		
	default	['0xedf206987be3a32111f16c0807c9055e2b8b8fc84f42768015cb7f8471137890, 100 eth', '0x0ca1573d73a070cfa5c48ddaf000b9480e94805f96a79ffa2d5bc6cc3288a92 100 eth', '0x2688eabfae4637b73752d342991579500f231c72d52dd22b29bf018c0df4b 100 eth', '0x4a4dfe519c6182638d18c75523a95ed55a938426d5e80ac55a39ed83f9e4c5 100 eth', '0x60fae350e15bdfdc227fc0616dbe26acb5f05d65d469a811383926a6759402 100 eth', '0x9085677b64cb52d4b36058be795cb315722a361fb78b042a02600bcb2b3f2 100 eth', '0x372f46eae3eb91865809a90339acea1697555021d583dceb7dd05a635de75 100 eth', '0x48d73da350f98b1b16ede5fab0078c1ee2c3525483d5365626b4ba3d79868 100 eth', '0x669fd08dd8760b47b368153b2d8483c08295a0fa2853684746bf84ea533a6 100 eth', '0x6d3f46df88ffbaf2c7c5a9567f6c26414fa205ae6ca27312a656115a71dfc9f4 100 eth']		
	items			
	•	type	*string*	
• **Server.BlockTime**	If not null, enable automatic mining with BlockTime interval, in seconds			
	anyOf	•	type	*number*
		•	type	*null*
• **Server.GasPrice**	Price of gas for the server			
	type	*integer*		
	default	20000000000		
• **Server.GasLimit**	Gas limit for the server			
	type	*integer*		
	default	6721975		
• **Client.Endpoint**	Endpoint to which the RPC client should connect to			
	type	*string*		
	default	http://127.0.0.1:8545		
• **Client.GasPrice**	Default gas price for transactions			
	anyOf	•	type	*integer*
		•	type	*null*
• **Client.GasLimit**	Default gas limit for the transactions			
	anyOf	•	type	*integer*
		•	type	*null*

Continued on next page

Table 1 – continued from previous page

• **Compiler.Optimize**	Solidity compiler optimize runs, or null for no optimization			
	anyOf	•	type	*integer*
		•	type	*null*
• **Linter.Plugins**	List of plugins for ethlint linter			
	type	*array*		
	default	[‘security’]		
	items			
	•	type	*string*	
• **Linter.Rules**	Rules (configuration) for ethlint linter			
	type	*object*		
	default	OrderedDict([(‘quotes’, [‘error’, ‘double’]), (‘indentation’, [‘error’, 4])])		
• **Testing.RunServer**	Run a server instance on creation of the testing context			
	type	*boolean*		
	default	True		
• **Testing.PortRange**	Port range that can be used by the tests			
	type	*array*		
	default	[8600, 8700]		
	items			
	•	type	*integer*	
		maximum	65535	
		minimum	1	
	maxItems	2		
	minItems	2		
additionalProperties	False			

CHAPTER 4

solitude module

class solitude.**Factory**(*cfg*)

Bases: `object`

Create a Factory object

The Factory object can be used to create pre-configured objects from a solitude configuration dictionary.

Parameters **cfg** – solitude configuration dictionary

__init__(*cfg*)

Initialize self. See help(type(self)) for accurate signature.

create_client(*endpoint=None*) → ETHClient

Create a ETHClient object, used to interact with an ethereum node.

Parameters **endpoint** – if set, it overrides the Client.Endpoint setting in the configuration

create_compiler() → Compiler

create_linter(*add_contract_dir=False*) → Linter

Create a Linter object, used to invoke solium.

Parameters **add_contract_dir** – whether to load contracts from the directory specified in Compiler.ContractDir or not

create_server() → ETHTestServer

Create a ETHTestServer object, used to start a ganache test node.

get_objectlist() → ContractObjectList

get_project_name()

get_required()

get_sourcelist() → ContractSourceList

solitude.**read_config_file**(*url: str*) → dict

Read a solitude configuration from YAML or JSON.

Parameters **`url`** – URL or path of configuration; if *url* ends with '.yaml', the configuration file is interpreted as YAML, otherwise it is assumed to be JSON.

Returns configuration dictionary

`solitude.`**`write_config_file`**`(`*cfg: dict*, *path: str*`)` → None

Write configuration dictionary to file. If the file exists, it will be overwritten.

Parameters

- **`cfg`** – solitude configuration dictionary.
- **`path`** – destination path.

`solitude.`**`make_default_config`**`()` → dict

Create a default solitude configuration.

Returns a configuration dictionary

CHAPTER 5

solitude.common module

class solitude.common.**FileMessage**

Bases: tuple

message (error, warning) related to a text file

column

column number, optional

line

line number, optional

message

the message string

type

a string indicating the message type

unitname

source unit name or file name

solitude.common.**file_message_format**(*m: solitude.common.structures.FileMessage*)

Format a FileMessage object to string

class solitude.common.**TransactionInfo**

Bases: tuple

Transaction information

address

contract instance address

contractname

contract name

fnargs

function arguments as tuple

function

function name

receipt
full web3 receipt

txargs
transaction arguments as dictionary ('gas', 'gasprice', 'value')

txhash
Alias for field number 6

unitname
source unit name

solitude.common.**hex_repr**(*b: bytes*, *pad: Optional[int] = None*, *prefix=True*)
Get hex string representation of a byte array

Parameters

- **b** – byte array
- **pad** – pad to fixed number of characters
- **prefix** – prefix with '0x'

solitude.common.**get_resource_path**(*resource_name: str*)
Get location of a solitude resource file in the filesystem

Parameters **resource_name** – name of the resource

Returns resource file path

solitude.common.**get_global_config**()
Get the solitude global configuration, containing global settings of the solitude framework.

Returns the solitude global config

solitude.common.**update_global_config**(*config: dict*)
Update the solitude global configuration from a dictionary

Parameters **config** – dictionary containing the values to replace

solitude.common.**copy_from_url**(*url: str*, *destination*, *decode=False*)
Copy file from URL to file-like

Parameters

- **url** – source URL (see `open_url()`)
- **destination** – destination writable file-like
- **decode** – interpret the stream as utf-8 text and convert it to string

solitude.common.**read_from_url**(*url: str*, *decode=False*)
Read file from URL to a byte array or string

Parameters

- **url** – source URL (see `open_url()`)
- **decode** – interpret the stream as utf-8 text and convert it to string

Returns a byte array (bytes) if decode is False, otherwise a string (str)

solitude.common.**open_url**(*url: str*, *decode=False*)
Open URL and return readable file-like

The URL can have one of the following schemas

- https://{hostname}/{path} - HTTPS url

- http://{hostname}/{path} - HTTP url
- resource://{name} - solitude resource, by name
- file://{path} file on the filesystem, by path

  Parameters

  - **`url`** – source URL
  - **`decode`** – interpret the stream as utf-8 text and convert it to string

  Returns a file-like object, binary if decode is False, otherwise text

`solitude.common.`**`read_config_file`**(*url: str*) → dict
Read a solitude configuration from YAML or JSON.

Parameters **`url`** – URL or path of configuration; if *url* ends with '.yaml', the configuration file is interpreted as YAML, otherwise it is assumed to be JSON.

Returns configuration dictionary

`solitude.common.`**`read_yaml_or_json`**(*url: str*) → dict
Read a YAML or JSON document.

Parameters **`url`** – URL or path; if *url* ends with '.yaml', the document is interpreted as YAML, otherwise it is assumed to be JSON.

Returns a dictionary with the document contents.

`solitude.common.`**`make_default_config`**() → dict
Create a default solitude configuration.

Returns a configuration dictionary

class `solitude.common.`**`ContractObjectList`**
Bases: `object`

A collection of compiled contracts

`__init__`()
Create an empty collection of compiled contracts

`add_contract`(*unitname: str*, *contractname: str*, *contract: dict*)
Add a contract, uniquely identified by (unitname, contractname).

Parameters

- **`unitname`** – source unit containing the contract
- **`contractname`** – name of the contract
- **`contract`** – contract data dictionary, as produced by the compiler module

`add_directory`(*path: str*) → None
Add all contracts from a directory.

Parameters **`path`** – path of the directory containing the contracts data.

`contracts`
All contracts, as a dictionary of (unitname, contractname) -> data

`find`(*suffix: Optional[str], contractname: str*) → List[Tuple[str, str]]
Find contracts by unitname suffix and full contractname.

Example: ("erc20/ERC20.sol", "ERC20") matches ("/home/user/contracts/erc20/ERC20.sol", "ERC20").

Parameters

- **suffix** – suffix to match the contract source unit name, or None; if it is None, any unit name is matched.
- **contractname** – full name of the contract

save_directory (*path: str*) → None

Save all contracts to a directory.

Parameters **path** – path of destination directory; the directory must exist.

select (*selector: str*) → dict

Find a single contract matching the contract selector string.

The selector string is a string in either of the following forms:

- **"{suffix}:{contractname}": source unit name suffix and contract name,** separated by ':'. Example: "erc20/ERC20.sol:ERC20" matches contract named "ERC20" in source unit "/home/user/contracts/erc20/ERC20.sol".
- **"{contractname}": only the contract name. Example: "ERC20" matches** contract named "ERC20".

If the selector matches multiple contract, this function will raise an exception of type ValueError.

Parameters **selector** – contract selector

update (*other: solitude.common.contract_objectlist.ContractObjectList*) → None

Add all contracts from other ContractObjectList.

Parameters **other** – other ContractObjectList with contracts to add

class solitude.common.**ContractSourceList**

Bases: object

A collection of contract sources

__init__ ()

Create an empty collection of contract sources

add_directory (*path: str, ext_filter: Optional[List[str]] = ['.sol']*) → None

Add all sources from a directory.

Parameters

- **path** – directory path
- **ext_filter** – list of allowed extensions for the source file names, including the '.' character (e.g. [".sol"]), or None for any extension

add_file (*path: str*) → None

Add file to the list of sources.

Parameters **path** – file path

add_files (*sources: List[str]*) → None

Add list of files to the list of sources.

Parameters **sources** – list of file paths

add_string (*unitname: str, source: str*) → None

Add a source string to the list of sources

Parameters

- **unitname** – a name for the provided source unit

- **`source`** – source code text

`file_sources`
All added file sources as list of paths

`text_sources`
All added source strings as dictionary of unitname -> source.

`solitude.common.`**`path_to_unitname`**(*path: str*) → str
Create a source unit name from a path

Parameters **`path`** – source file path

Returns the corresponding normalized source unit name

`class` `solitude.common.`**`Dump`**(*filename: str = None*, *fileobj=None*, *prefix: str = None*)
Bases: `object`

`FLUSH = 1000`

`__init__`(*filename: str = None*, *fileobj=None*, *prefix: str = None*)
Initialize self. See help(type(self)) for accurate signature.

`close`()

`push`(*name*)

`raw`(*msg*, **args*)

`write`(*msg*, **args*)

`class` `solitude.common.`**`RPCClient`**(*endpoint: str*)
Bases: `object`

Communicate with a JSON-RPC server

Any method can be called by RPCClient.rpcFunctionName(arguments…)

`__init__`(*endpoint: str*)

Parameters **`endpoint`** – JSON-RPC server URL

`batch_call`(*functions: List[Tuple[str, list]]*)
Perform a batch call

Parameters **`function`** – list of the requests to perform in batch, as tuples of (method name, list of arguments)

Returns the list of responses from the server

CHAPTER 6

solitude.testing module

solitude.testing.**SOL_new**(*cfg: Union[dict, str] = 'solitude.yaml', relative_to: Optional[str] = None*) → solitude.testing.context.TestingContext

Create a new testing context

Parameters

- **cfg** – configuration dictionary or path. If *cfg* is a string, it is interpreted as a path to the yaml or json file containing the configuration dictionary.
- **relative_to** – a path, or None; if *cfg* is a path and *relative_to* is not None, make the path of the configuration file *cfg* relative to the parent directory of *relative_to*. This can be used with *__file__* to make the configuration file location relative to the test script.

solitude.testing.**SOL**

alias of `solitude.testing.context.TestingContext`

class solitude.testing.**TestingContext**(*cfg: dict*)

Bases: `object`

__init__(*cfg: dict*)

Create a testing context containing configured instances of the client, server and compiler.

Contracts from Project.ObjectDir (if not null) are added to the client's collection.

A server is started if Testing.RunServer is true. In this case, the client is connected to the new server endpoint address, whatever it is, overriding the client endpoint configuration.

Parameters **cfg** – configuration dictionary

account(*address*)

Enter a context which uses a specific account to perform all transactions in the context.

Parameters **address** – address of the account to use

Returns an account context

Contexts can be nested. In this case, the account in the last context will be used.

```
with client.account(client.address(0)):
    client.deploy("ContractName", args=())
```

address (*account_id: int*)
Get the address of an account in the ETH node

Parameters **account_id** – index of the account in the ETH node

Returns address of the account

capture (*pattern*)
Enter a context which captures events emitted by transactions.

Parameters **pattern** – a glob pattern string, or a regex object, to match the event name

Returns a capture context

The event name is in the format: `{unitname}:{contractname}.{eventname}`

All emitted events that match are stored within the client and can be accessed with `client.get_events()`. They are cleared before every capture.

Nested captures will filter events that match any of the patterns in the context.

```
with client.capture("*:MyToken.Transfer"):
    my_token_instance.transfer(address, 13)

assert client.get_events()[0].args[2] == 13

with client.capture(re.compile(r".*:MyToken\.Transfer")):
    my_token_instance.transfer(address, 1)

assert client.get_events()[0].args[2] == 1
```

cfg
Configuration

clear_events () → None
Clear events generated within the last capture context

client
Client instance

compiler
Compiler instance

deploy (*contract_selector: str*, *args=()*, *wrapper=<class 'solitude.client.contract.ContractBase'>*)
Deploy a contract

Parameters

- **contract_selector** – contract selector string, see *solitude.common.ContractObjectList.select()*. The contract must be present in the compiler's collection and must contain ABI and bytecode.
- **args** – constructor arguments
- **wrapper** – wrapper class for contract (see ContractBase)

get_accounts (*reload=False*) → list
Get the accounts stored in the ETH node

Parameters **reload** – whether to refresh the account list by querying the node

Retrurn list of accounts

`get_current_account``()`
Get the account which is currently in use

Returns address of the account in use

`get_events``()` → List[solitude.client.eth_client.EventLog]
Get events generated within the last capture context

Returns list of event logs

`get_last_blocktime``()` → int
Get timestamp of last mined block

Returns last block's timestamp (in seconds)

`increase_blocktime_offset``(`*seconds: int*`)` → int
Increase the offset to apply to block.timestamp for newly mined blocks

Parameters **`seconds`** – number of seconds to add to block.timestamp offset (in seconds)

Returns new block.timestamp offset (in seconds)

`mine_block``()` → None
Ask the ETH node to mine a new block

`server`
Server instance

`teardown``()`
Teardown the testing context, terminating the test server if any.

`solitude.testing.`**`sol`**`()`
pytest fixture for a testing context configured with the default configuration file, solitude.yaml.

CHAPTER 7

solitude.server module

class solitude.server.**ETHTestServer**(*executable='ganache-cli', host='127.0.0.1', port: int = 8545, accounts: List[Tuple[str, int]] = None, blocktime: Optional[float] = None, gasprice=20000000000, gaslimit=6721975*)

Bases: object

Wrapper around the ganache-cli executable

__init__(*executable='ganache-cli', host='127.0.0.1', port: int = 8545, accounts: List[Tuple[str, int]] = None, blocktime: Optional[float] = None, gasprice=20000000000, gaslimit=6721975*)

Create a ganache-cli server instance

Parameters

- **executable** – path to the ganache-cli executable file
- **host** – address of the interface to which the server will bind to
- **port** – port on which the server will listen
- **accounts** – list of accounts to create on the server, as a list of (private_key, wei_balance) tuples, where private_key is a hex string of 32 bytes prefixed with "0x".
- **blocktime** – if not None, enable automatic mining with blocktime interval, in seconds.
- **gasprice** – price of gas (wei)
- **gaslimit** – gas limit

endpoint

Endpoint URL

is_alive() → bool

Check if the ganache-cli process is running

Returns True if ganache-cli is running

kill(*timeout: float = 1.0*) → None

Forcibly kill (SIGKILL) the ganache-cli process and wait

Parameters **`timeout`** – time to wait for ganache-cli to terminate

`start`(*timeout: float = 15.0*) → None

Start ganache-cli in the background.

When this function terminates (without errors), it means the server is running in the background and ready to receive requests.

Parameters **`timeout`** – timeout to wait for ganache-cli to respond, seconds

`stop`(*timeout: float = 15.0*) → None

Terminate (SIGTERM) the ganache-cli process and wait. If this fails, kill the process (SIGKILL).

Parameters **`timeout`** – time to wait for ganache-cli to terminate

`solitude.server.`**`kill_all_servers`**`()`

CHAPTER 8

solitude.client module

class solitude.client.**ETHClient**(*endpoint: str*)

Bases: solitude.client.eth_client.AccountContext, solitude.client.eth_client.EventCaptureContext

The ethereum node client object allows to communicate with an ethereum node.

It is mainly used to produce contract objects which allow to interact with a contract instance on the blockchain.

It stores a collection of contracts, their ABI and optionally their bytecode.

__init__(*endpoint: str*)

Initialize a new ETH client without any contract.

Parameters **endpoint** – URL of the ethereum server node

account(*address*)

Enter a context which uses a specific account to perform all transactions in the context.

Parameters **address** – address of the account to use

Returns an account context

Contexts can be nested. In this case, the account in the last context will be used.

```
with client.account(client.address(0)):
    client.deploy("ContractName", args=())
```

add_filter(*contracts: List[solitude.client.contract.ContractBase], event_names: List[str], parameters=None*) → solitude.client.eth_client.Filter

Subscribe to events occurring on the ETH node

Creates a filter on the ETH node. Returns an object with the filter information, which can be used to retrieve the events or unsubscribe.

Parameters

- **contracts** – list of contract instances which can generate the event. All instances must refer to the same contract, possibly deployed at multiple addresses.

- **event_names** – names of events to listen for
- **parameters** – additional raw topics (optional)

Returns a Filter object

address (*account_id: int*)
Get the address of an account in the ETH node

Parameters **account_id** – index of the account in the ETH node

Returns address of the account

capture (*pattern*)
Enter a context which captures events emitted by transactions.

Parameters **pattern** – a glob pattern string, or a regex object, to match the event name

Returns a capture context

The event name is in the format: `{unitname}:{contractname}.{eventname}`

All emitted events that match are stored within the client and can be accessed with `client.get_events()`. They are cleared before every capture.

Nested captures will filter events that match any of the patterns in the context.

```
with client.capture("*:MyToken.Transfer"):
    my_token_instance.transfer(address, 13)

assert client.get_events()[0].args[2] == 13

with client.capture(re.compile(r".*:MyToken\.Transfer")):
    my_token_instance.transfer(address, 1)

assert client.get_events()[0].args[2] == 1
```

clear_events () → None
Clear events generated within the last capture context

contracts
The collection of all contracts known by this client, as a ContractObjectList object

deploy (*contract_selector: str*, *args=()*, *wrapper=<class 'solitude.client.contract.ContractBase'>*)
Deploy a contract

Parameters

- **contract_selector** – contract selector string, see *solitude.common.ContractObjectList.select()*. The contract must be present in the compiler's collection and must contain ABI and bytecode.
- **args** – constructor arguments
- **wrapper** – wrapper class for contract (see ContractBase)

get_accounts (*reload=False*) → list
Get the accounts stored in the ETH node

Parameters **reload** – whether to refresh the account list by querying the node

Retrurn list of accounts

get_current_account ()
Get the account which is currently in use

Returns address of the account in use

get_events() → List[solitude.client.eth_client.EventLog]
Get events generated within the last capture context

Returns list of event logs

get_last_blocktime() → int
Get timestamp of last mined block

Returns last block's timestamp (in seconds)

import_raw_key(*private_key: str*, *passphrase: str = ”*)

increase_blocktime_offset(*seconds: int*) → int
Increase the offset to apply to block.timestamp for newly mined blocks

Parameters **seconds** – number of seconds to add to block.timestamp offset (in seconds)

Returns new block.timestamp offset (in seconds)

iter_filters(*filters: List[solitude.client.eth_client.Filter], interval=1.0*)
Iterate over events generated by a list of filters

Parameters **interval** – polling interval in seconds

Returns an iterator of EventLog objects

mine_block() → None
Ask the ETH node to mine a new block

miner_start(*num_threads: int*)

remove_filter(*flt: solitude.client.eth_client.Filter*) → None
Unsubscribe from previously created filter.

Clean up the filter from the ETH node.

Parameters **flt** – a Filter object (created by `EthClient.add_filter()`)

rpc
A raw JSON-RPC client instance to communicate with the ETH node

set_default_gaslimit(*gas: Optional[int]*)
Set the default gas limit for transactions

Parameters **gas** – default gas limit, or None. If the gas limit is not set either through the default or explicitly in the transaction, web3 will call eth.estimateGas first to determine this value.

set_default_gasprice(*gasprice: Optional[int]*)
Set the default gas price for transactions

Parameters **gasprice** – default gas limit, or None. If the gas price is not set, web3 will call eth.gasPrice first to determine this value.

unlock_account(*address: str*, *passphrase: str = ”*, *unlock_duration: int = 300*)

update_contracts(*contracts: solitude.common.contract_objectlist.ContractObjectList*)
Update the collection of contracts known to this client

Parameters **contracts** – a collection of contracts (see ContractObjectList)

use(*contract_selector: str*, *address: str*, *wrapper=<class 'solitude.client.contract.ContractBase'>*)
Use a contract at a specific address

Parameters

- **contract_selector** – contract selector string, see *solitude.common.ContractObjectList.select()*. The contract must be present in the client's collection and must contain the ABI at least.
- **args** – constructor arguments
- **account** – deployer account, default is account 0
- **wrapper** – wrapper class for contract (see ContractBase)

web3
A raw web3 library client instance connected to the ETH node

class solitude.client.**BatchCaller**(*client: solitude.client.eth_client.ETHClient*)
Bases: object

Utility to batch function call requests to the ETH node

__init__(*client: solitude.client.eth_client.ETHClient*)
Create a BatchCaller

Parameters **client** – an ETH client

add_call(*contract: solitude.client.contract.ContractBase*, *func: str*, *args=()*) → None
Add a function call to the batch call

Parameters

- **contract** – a contract object (from EthClient.deploy() or EthClient.use()).
- **func** – function name
- **args** – function arguments

execute() → list
Execute the call batch

Returns a list containing the result from each function call, in the same order in which they were added.

class solitude.client.**Filter**
Bases: tuple

Filter information

contractname
Contract which contains the event definition

event_names
Names of the events to filter

index
Index of the filter on the ETH node

unitname
Source unit of the contract which contains the event definition

valid
Whether the filter is still valid and it should keep being used

class solitude.client.**EventLog**
Bases: tuple

Event information

address
Address of the contract instance which produced the event

args
Arguments of the emitted event

contractname
Contract which contains the event definition

data
Raw event data from web3

name
Event name

unitname
Source unit of the contract which contains the event definition

class solitude.client.**ContractBase**(*client: solitude.client.eth_client.ETHClient*, *unitname: str*, *contractname: str*, *contract: web3.contract.Contract*)
Bases: `object`

Wrapper around web3 contract object. Allows to define wrapper methods to call contract functions

__init__(*client: solitude.client.eth_client.ETHClient*, *unitname: str*, *contractname: str*, *contract: web3.contract.Contract*)

Parameters

- **client** – solitude client object which produced this instance
- **unitname** – name of the source unit containing the contract
- **contractname** – name of the contract
- **contract** – web3 contract instance:

abi
Contract ABI

account
Account which is being used as sender

address
Contract address

call(*func: str*, **args*)
Call a function in the contract

Parameters

- **func** – function name
- ***args** – function arguments

functions
Functions from web3 contract object

name
Contract name

transact_sync(*func: str*, **args*, *value: int = None*, *gas: int = None*, *gasprice: int = None*) → solitude.common.structures.TransactionInfo
Send a transaction and wait for its receipt

Parameters

- **func** – function name
- ***args** – function arguments
- **value** – optional amount of ether to send (in wei)
- **gas** – optional gas limit
- **gasprice** – optional gas price

Returns transaction information

unitname
Name of the source unit containing this contract

web3
Raw web3 contract object

CHAPTER 9

solitude.debugger module

class solitude.debugger.**EvmTrace**(*rpc: solitude.common.rpc_client.RPCClient*, *contracts: solitude.common.contract_objectlist.ContractObjectList*)

Bases: `object`

Access debug information from the ETH server

`__init__`(*rpc: solitude.common.rpc_client.RPCClient*, *contracts: solitude.common.contract_objectlist.ContractObjectList*)

Create an EvmTrace instance

Parameters

- **`rpc`** – RPC client connected to the ETH server
- **`contracts`** – a collection of contracts (see ContractObjectList)

`trace_iter`(*txhash: bytes*) → Iterator[Tuple[solitude.debugger.evm_trace.TraceStep, solitude.debugger.evm_trace.CallStackEvent]]

Iterate contract execution steps (instructions)

Parameters **`txhash`** – transaction hash to inspect, as byte array

Returns generator of tuples of (TraceStep, CallStackEvent)

class solitude.debugger.**TraceStep**

Bases: `tuple`

Debugger step (instruction) information

`code`
a SourceMapping object containing the source code and line information

`contractname`
contract name

`depth`
call stack depth

`error`
Error message

`fileno`
index which identifies the source unit

`gas`
Gas cost of the instruction

`index`
incrementing index of the step

`jumptype`
type of jump, *'i'* for 'jump into call', *'o'*, for 'jump out of call', or empty (*''*)

`length`
length of the source code mapped to this instruction

`memory`
EVM memory as list of hex strings

`op`
opcode string

`pc`
program counter

`stack`
EVM stack as list of hex strings

`start`
index of the character in the source file where the source code mapped to this instruction starts

`storage`
EVM storage as dictionary of hex strings

class `solitude.debugger.`**`SourceMapping`**
Bases: `tuple`

Source code and line information related to an instruction

`line_index`
line index where the relevant portion begins

`line_pos`
index of the column where the relevant portion begins (in line)

`line_start`
index of the character where the line starts in the file

`lines`
full source text split in lines

`source`
full source text

`unitname`
source unit name

class `solitude.debugger.`**`CallStackElement`**
Bases: `tuple`

Basic stack frame information

`prev`
the TraceStep before entering a call

step
the TraceStep after entering a call

class solitude.debugger.**CallStackEvent**
Bases: tuple

Call stack event information

data
Event data. If the event is of type *'push'*, a *CallStackElement*

event
Type of event. Can be *'push'*, *'pop'* or *None*

class solitude.debugger.**EvmDebugCore**(*client: solitude.client.eth_client.ETHClient*, *txhash: bytes*, *windowsize=50*)
Bases: object

Provides common debugger-like access to the EVM's debug information

INVALID_STEP = <solitude.debugger.evm_debug_core.Step object>

__init__(*client: solitude.client.eth_client.ETHClient*, *txhash: bytes*, *windowsize=50*)
Create an EvmDebugCore.

Parameters

- **client** – an *ETHClient* connected to the ETH node
- **txhash** – transaction hash, as bytes
- **windowsize** – amount of previous and next steps buffered, for a total of previous (windowsize) + current (1) + next (windowsize).

get_callstack_depth() → int
Get the call stack depth :return: number of frames in the call stack

get_frames() → List[solitude.debugger.evm_debug_core.Frame]
Get call stack frames :return: a list of *Frame*

get_step(*offset=0*) → solitude.debugger.evm_debug_core.Step
Get step, relative to current step.

Parameters **offset** – step offset, relative to the current one. Can be in range (-windowsize, windowsize), according to the windowsize value provided in the constructor.

Returns a *Step*

get_values() → Dict[str, solitude.debugger.evm_debug_core.Value]
Get named values in the current step, from function parameters and local variables.

Returns list of *Value*

step()
Step one instruction forward

class solitude.debugger.**Function**(*name: str, parameters: List[solitude.debugger.evm_debug_core.Value]*)
Bases: solitude._internal.oi_serializable.ISerializable

Object containing a function call information

__init__(*name: str, parameters: List[solitude.debugger.evm_debug_core.Value]*)
Create a Function object

Parameters

- **name** – function name

- **`parameters`** – list of function parameters, as `Value` objects

`static from_obj`(*obj*)

`to_obj`()

class `solitude.debugger.`**`Frame`**(*prev: solitude.debugger.evm_trace.TraceStep*, *cur: solitude.debugger.evm_trace.TraceStep*)

Bases: `solitude._internal.oi_serializable.ISerializable`

Call stack frame information

Variables

- **`locals`** – dictionary of local variable values
- **`return_values`** – list of values produced by return statements
- **`function`** – function call information

Step information is lost during serialization, and the three attributes above are kept

`__init__`(*prev: solitude.debugger.evm_trace.TraceStep*, *cur: solitude.debugger.evm_trace.TraceStep*)

Create a Frame object

Parameters

- **`prev`** – step before entering the function
- **`cur`** – step after entering the function

`static from_obj`(*obj*)

`to_obj`()

class `solitude.debugger.`**`Step`**(*step: Optional[solitude.debugger.evm_trace.TraceStep], event: Optional[solitude.debugger.evm_trace.CallStackEvent]*)

Bases: `object`

Single instruction step information

Variables

- **`ast`** – AST nodes mapped to the instruction, as dictionary of (node type name -> node dict)
- **`values`** – values associated to this instruction (variable assignment, value produced by evaluation of statement, . . .)

`__init__`(*step: Optional[solitude.debugger.evm_trace.TraceStep], event: Optional[solitude.debugger.evm_trace.CallStackEvent]*)

Create a Step object

Parameters

- **`step`** – step information
- **`event`** – call stack event associated with the step

This object may be create empty, with null step and event data.

`valid`

Wether this object contains step information or is empty

Returns True if not empty, otherwise False

class `solitude.debugger.`**`Value`**(*vtype: str*, *name: str*, *value*, *kind: str*, *origin=None*)

Bases: `solitude._internal.oi_serializable.ISerializable`

Value debug information

It represents a value associated to a named entity in the source code. Only supports numeric values.

`__init__`(*vtype: str*, *name: str*, *value*, *kind: str*, *origin=None*)
Create a Value object

Parameters

- **`vtype`** – value type name
- **`name`** – value name
- **`value`** – integer content of value
- **`kind`** – one of ValueKind enum values
- **`origin`** – type of AST node from which the variable information was extracted

`static from_obj`(*obj*)

`to_obj`()

`value_repr`() → str
Get string representation of the value

Returns string representation

`class` `solitude.debugger.`**`InteractiveDebuggerOI`**(*txhash*, *client*, *code_lines=(3, 6)*)
Bases: `solitude._internal.oi_interface.ObjectInterface`

`__init__`(*txhash*, *client*, *code_lines=(3, 6)*)
Initialize self. See help(type(self)) for accurate signature.

`cmd_backtrace`(*args*)

`cmd_break`(*args*)

`cmd_continue`(*args*)

`cmd_delete`(*args*)

`cmd_finish`(*args*)

`cmd_frame`(*args*)

`cmd_info_args`(*args*)

`cmd_info_breakpoints`(*args*)

`cmd_info_locals`(*args*)

`cmd_list`(*args*)

`cmd_next`(*args*)

`cmd_print`(*args*)

`cmd_quit`(*args*)

`cmd_step`(*args*)

`cmd_stepi`(*args*)

`format_code`(*step*, *before=None*, *after=None*)

`static get_source_lines`(*step: solitude.debugger.evm_trace.TraceStep*, *strip=False*, *color='green'*, *before=0*, *after=0*) → solitude._internal.oi_common_objects.ColorText

`on_breakpoint`(*args*)

on_revert(*args*)

on_step(*args*)

on_terminate(*args*)

CHAPTER 10

solitude.compiler module

class solitude.compiler.**Compiler**(*executable: str*, *optimize: Optional[int] = None*)

Bases: `object`

Wrapper for the solidity contract compiler

__init__(*executable: str*, *optimize: Optional[int] = None*)

Create a compiler instance

Parameters

- **executable** – path to compiler executable binary
- **optimize** – solidity optimizer runs, or None

compile(*sourcelist: solitude.common.contract_sourcelist.ContractSourceList*) → solitude.common.contract_objectlist.ContractObjectList

Compile all contracts in a collection of sources

Parameters **sourcelist** – collection of sources as ContractSourceList

Returns compiled contracts as ContractObjectList

CHAPTER 11

solitude.linter module

class `solitude.linter.`**`Linter`**(*executable: str, plugins: List[str], rules: dict, parallelism: int = 4*)

Bases: `object`

The linter object allows linting groups of contracts.

`__init__`(*executable: str, plugins: List[str], rules: dict, parallelism: int = 4*)

Create a Linter

Parameters

- **`executable`** – path to the solium executable file
- **`plugins`** – list of solium plugins
- **`rules`** – dictionary containing solium rules
- **`parallelism`** – maximum number of parallel instances to be run

`lint`(*sourcelist: solitude.common.contract_sourcelist.ContractSourceList*) → Iterator[Tuple[str, solitude.common.structures.FileMessage]]

Lint a group of contracts

Parameters **`sourcelist`** – source files to lint, as ContractSourceList

Returns an iterator of FileMessage objects containing the linter output information

CHAPTER 12

solitude.tools module

class solitude.tools.**Tool**(*tooldir: str*, *name: str*, *version: str*)

Bases: `object`

An external tool that can be installed on the local filesystem and used

__init__(*tooldir: str*, *name: str*, *version: str*)

Initialize self. See help(type(self)) for accurate signature.

add()

Install the tool into the tools directory

get(*key: str*) → str

Get a module from the tool

Parameters **key** – a string key (name) associated with the module, usually the name of the file.

Returns the filesystem path of the module location

have() → bool

Check if the tool is present in the tools directory

name

Tool name

provided

Get the provided modules

Returns dict of (key -> path) of all modules provided by the tool

remove()

Remove (delete) the tool from the tools directory

version

Tool version string

solitude.tools.**Solc**

alias of `solitude.tools.solc.SolcNativeLinux`

class `solitude.tools.`**`GanacheCli`**(*tooldir: str*, *version: str*)

Bases: `solitude.tools.base.ToolNpmTemplate`

`__init__`(*tooldir: str*, *version: str*)

Initialize self. See help(type(self)) for accurate signature.

class `solitude.tools.`**`EthLint`**(*tooldir: str*, *version: str*)

Bases: `solitude.tools.base.ToolNpmTemplate`

`__init__`(*tooldir: str*, *version: str*)

Initialize self. See help(type(self)) for accurate signature.

Python Module Index

Index