crytic-compile

Release 0.1.6

Mar 23, 2020

Contents:

CHAPTER 1

crytic_compile

1.1 crytic_compile package

1.1.1 Subpackages

1.1.1.1 crytic_compile.compiler package

Submodules

crytic_compile.compiler.compiler module

Handle the compiler version

class crytic_compile.compiler.compiler.**CompilerVersion**(*compiler*, *version*, *optimized*)

Bases: tuple

compiler
Alias for field number 0

optimized
Alias for field number 2

version
Alias for field number 1

Module contents

1.1.1.2 crytic_compile.cryticparser package

Submodules

crytic_compile.cryticparser.cryticparser module

Module handling the cli arguments

crytic_compile.cryticparser.cryticparser.**init**(*parser: argparse.ArgumentParser*)
Add crytic-compile arguments to the parser

Parameters **parser** –

Returns

crytic_compile.cryticparser.defaults module

Default value for options

Module contents

Init module

1.1.1.3 crytic_compile.platform package

Submodules

crytic_compile.platform.abstract_platform module

Abstract Platform

class crytic_compile.platform.abstract_platform.**AbstractPlatform**(*target: str*, ***kwargs*)
Bases: object

This is the abstract class for the platform

HIDE = False

NAME = ''

PROJECT_URL = ''

TYPE = 0

compile(*crytic_compile: CryticCompile*, ***kwargs*)
Run the compilation

Parameters

- **crytic_compile** –
- **kwargs** –

Returns

guessed_tests() → List[str]
Guess the potential unit tests commands

Returns list of unit tests command guessed

is_dependency(*path: str*) → bool
Check if the target is a dependency

Parameters **path** –

Returns

static is_supported(*target: str*, ***kwargs*) → bool
Check if the target is a project supported by this platform

Parameters **target** –

Returns

platform_name_used
Return the underlying platform used

Returns

platform_project_url_used
Return the underlying platform url used

Returns

platform_type_used
Return the underlying platform url used

Returns

target
Return the target name

Returns

exception crytic_compile.platform.abstract_platform.**IncorrectPlatformInitialization**
Bases: Exception

Exception raises if a platform was not properly defined

crytic_compile.platform.all_export module

Module containing all the supported export functions

crytic_compile.platform.all_platforms module

Module containing all the platforms

crytic_compile.platform.archive module

Archive platform. It is similar to the standard platform, except that the file generated contains a "source_content" field
Which is a map: filename -> sourcecode

class crytic_compile.platform.archive.**Archive**(*target: str*, ***kwargs*)
Bases: *crytic_compile.platform.abstract_platform.AbstractPlatform*

Archive platform. It is similar to the Standard platform, but contains also the source code

HIDE = True

NAME = 'Archive'

PROJECT_URL = 'https://github.com/crytic/crytic-compile'

TYPE = 101

compile(*crytic_compile: CryticCompile*, ***_kwargs*)
Compile

Parameters

- **crytic_compile** –
- **_kwargs** –

Returns

is_dependency(*_path: str*) → bool
Check if the _path is a dependency. Always false

Parameters **_path** –

Returns

static is_supported(*target: str*, ***kwargs*) → bool
Check if the target is an archive

Parameters **target** –

Returns

crytic_compile.platform.archive.**export_to_archive**(*crytic_compile: CryticCompile*, ***kwargs*) → str
Export the archive

Parameters

- **crytic_compile** –
- **kwargs** –

Returns

crytic_compile.platform.archive.**generate_archive_export**(*crytic_compile: CryticCompile*) → Tuple[Dict[KT, VT], str]
Generate the archive export

Parameters **crytic_compile** –

Returns

crytic_compile.platform.brownie module

Brownie platform. https://github.com/iamdefinitelyahuman/brownie

class crytic_compile.platform.brownie.**Brownie**(*target: str*, ***kwargs*)
Bases: *crytic_compile.platform.abstract_platform.AbstractPlatform*

Brownie class

NAME = 'Brownie'

PROJECT_URL = 'https://github.com/iamdefinitelyahuman/brownie'

TYPE = 9

compile(*crytic_compile: CryticCompile*, ***kwargs*)
Compile the target

Parameters

- **crytic_compile** –
- **target** –
- **kwargs** –

Returns

is_dependency(*_path: str*) → bool
Check if the path is a dependency

Parameters **_path** –

Returns

static is_supported(*target: str*, ***kwargs*) → bool
Check if the target is a brownie env

Parameters **target** –

Returns

crytic_compile.platform.dapp module

Dapp platform. https://github.com/dapphub/dapptools

class crytic_compile.platform.dapp.**Dapp**(*target: str*, ***kwargs*)
Bases: *crytic_compile.platform.abstract_platform.AbstractPlatform*

Dapp class

NAME = 'Dapp'

PROJECT_URL = 'https://github.com/dapphub/dapptools'

TYPE = 4

compile(*crytic_compile: CryticCompile*, ***kwargs*)
Compile the target

Parameters

- **crytic_compile** –
- **target** –
- **kwargs** –

Returns

is_dependency(*path: str*) → bool
Check if the path is a dependency

Parameters **path** –

Returns

static is_supported(*target: str*, ***kwargs*) → bool
Heuristic used: check if "dapp build" is present in Makefile

Parameters **target** –

Returns

crytic_compile.platform.embark module

Embark platform. https://github.com/embark-framework/embark

class crytic_compile.platform.embark.**Embark**(*target: str*, ***kwargs*)
Bases: *crytic_compile.platform.abstract_platform.AbstractPlatform*

Embark platform

NAME = 'Embark'

PROJECT_URL = 'https://github.com/embarklabs/embark'

TYPE = 3

compile(*crytic_compile: CryticCompile*, ***kwargs*)
Compile the target

Parameters

- **crytic_compile** –
- **target** –
- **kwargs** –

Returns

is_dependency(*path: str*) → bool
Check if the path is a dependency

Parameters **path** –

Returns

static is_supported(*target: str*, ***kwargs*) → bool
Check if the target is an embark project

Parameters **target** –

Returns

crytic_compile.platform.etherlime module

Etherlime platform. https://github.com/LimeChain/etherlime

class crytic_compile.platform.etherlime.**Etherlime**(*target: str*, ***kwargs*)
Bases: *crytic_compile.platform.abstract_platform.AbstractPlatform*

Etherlime platform

NAME = 'Etherlime'

PROJECT_URL = 'https://github.com/LimeChain/etherlime'

TYPE = 5

compile(*crytic_compile: CryticCompile*, ***kwargs*)
Compile the target

Parameters

- **crytic_compile** –
- **target** –

- **kwargs –

Returns

is_dependency(*path: str*) → bool
Check if the path is a dependency

Parameters **path** –

Returns

static is_supported(*target: str*, ***kwargs*) → bool
Check if the target is an etherlime project

Parameters **target** –

Returns

crytic_compile.platform.etherscan module

Etherscan platform.

class crytic_compile.platform.etherscan.**Etherscan**(*target: str*, ***kwargs*)
Bases: *crytic_compile.platform.abstract_platform.AbstractPlatform*

Etherscan platform

NAME = 'Etherscan'

PROJECT_URL = 'https://etherscan.io/'

TYPE = 6

compile(*crytic_compile: CryticCompile*, ***kwargs*)
Compile the tharget :param crytic_compile: :param target: :param kwargs: :return:

is_dependency(*_path: str*) → bool
Always return false

Parameters **_path** –

Returns

static is_supported(*target: str*, ***kwargs*) → bool
Check if the target is an etherscan address

Parameters **target** –

Returns

crytic_compile.platform.etherscan.**convert_version**(*version: str*) → str
Convert the compiler version :param version: :return:

crytic_compile.platform.exceptions module

Crytic Compile Exceptions

exception crytic_compile.platform.exceptions.**InvalidCompilation**
Bases: Exception

Invalid compilation exception

crytic_compile.platform.solc module

Solc platform

class crytic_compile.platform.solc.**Solc**(*target: str*, ***kwargs*)
Bases: *crytic_compile.platform.abstract_platform.AbstractPlatform*

Solc platform

NAME = 'solc'

PROJECT_URL = 'https://github.com/ethereum/solidity'

TYPE = 1

compile(*crytic_compile: CryticCompile*, ***kwargs*)
Compile the target

Parameters

- **crytic_compile** –
- **kwargs** –

Returns

is_dependency(*_path: str*) → bool
Always return false

Parameters **_path** –

Returns

static is_supported(*target: str*, ***kwargs*) → bool
Check if the target is a solc project

Parameters **target** –

Returns

crytic_compile.platform.solc.**export_to_solc**(*crytic_compile: CryticCompile*, ***kwargs*) → Optional[str]
Export the project to the solc format

Parameters

- **crytic_compile** –
- **kwargs** –

Returns

crytic_compile.platform.solc.**get_version**(*solc: str*) → str
Get the compiler verison used

Parameters **solc** –

Returns

crytic_compile.platform.solc.**is_optimized**(*solc_arguments: str*) → bool
Check if optimization are used

Parameters **solc_arguments** –

Returns

crytic_compile.platform.solc.**relative_to_short**(*relative*)
Convert relative to short

Parameters **relative** –

Returns

crytic_compile.platform.solc_standard_json module

Handle compilation through the standard solc json format

class crytic_compile.platform.solc_standard_json.**SolcStandardJson**(*target: Union[str, dict] = None*, ***kwargs*)

Bases: *crytic_compile.platform.solc.Solc*

Represent the Standard solc Json object

NAME = 'Solc-json'

PROJECT_URL = 'https://solidity.readthedocs.io/en/latest/using-the-compiler.html#compi

TYPE = 10

add_remapping(*remapping: str*)
Append our remappings

Parameters **remapping** –

Returns

add_source_file(*file_path: str*)
Append file

Parameters **file_path** –

Returns

compile(*crytic_compile: CryticCompile*, ***kwargs*)
Compile the target

Parameters

- **crytic_compile** –
- **target** –
- **kwargs** –

Returns

to_dict() → Dict[KT, VT]
Patch in our desired output types

Returns

crytic_compile.platform.standard module

Standard crytic-compile export

class crytic_compile.platform.standard.**Standard**(*target: str*, ***kwargs*)

Bases: *crytic_compile.platform.abstract_platform.AbstractPlatform*

Standard platform (crytic-compile specific)

HIDE = True

NAME = 'Standard'

PROJECT_URL = 'https://github.com/crytic/crytic-compile'

TYPE = 100

compile(*crytic_compile: CryticCompile*, ***_kwargs*)
Compile the target (load file)

Parameters

- **crytic_compile** –
- **target** –
- **kwargs** –

Returns

is_dependency(*path: str*) → bool
Always return False

Parameters **path** –

Returns

static is_supported(*target: str*, ***kwargs*) → bool
Check if the target is the standard crytic compile export

Parameters **target** –

Returns

platform_name_used
Return the underlying platform used

Returns

platform_project_url_used
Return the underlying platform url used

Returns

platform_type_used
Return the underlying platform url used

Returns

crytic_compile.platform.standard.**export_to_standard**(*crytic_compile: CryticCompile*, ***kwargs*) → str
Export the project to the standard crytic compile format :param crytic_compile: :param kwargs: :return:

crytic_compile.platform.standard.**generate_standard_export**(*crytic_compile: CryticCompile*) → Dict[KT, VT]
Export the standard crytic compile export

Parameters **crytic_compile** –

Returns

crytic_compile.platform.standard.**load_from_compile**(*crytic_compile: CryticCompile, loaded_json: Dict[KT, VT]*) → Tuple[int, List[str]]
Load from json

Parameters

- **crytic_compile** –
- **loaded_json** –

Returns

crytic_compile.platform.truffle module

Truffle platform

class crytic_compile.platform.truffle.**Truffle**(*target: str*, ***kwargs*)
Bases: *crytic_compile.platform.abstract_platform.AbstractPlatform*

Truffle platform

NAME = 'Truffle'

PROJECT_URL = 'https://github.com/trufflesuite/truffle'

TYPE = 2

compile(*crytic_compile: CryticCompile*, ***kwargs*)
Compile the target

Parameters **kwargs** –

Returns

is_dependency(*path: str*) → bool
Check if the target is a dependency

Parameters **path** –

Returns

static is_supported(*target: str*, ***kwargs*) → bool
Check if the target is a truffle project

Parameters **target** –

Returns

crytic_compile.platform.truffle.**export_to_truffle**(*crytic_compile: CryticCompile*, ***kwargs*) → Optional[str]
Export to the truffle format

Parameters

- **crytic_compile** –
- **kwargs** –

Returns

crytic_compile.platform.types module

Handle the platform type

class crytic_compile.platform.types.**Type**
Bases: enum.IntEnum

Represent the different platform

ARCHIVE = 101

BROWNIE = 9

DAPP = 4

EMBARK = 3

ETHERLIME = 5

ETHERSCAN = 6

NOT_IMPLEMENTED = 0

SOLC = 1

SOLC_STANDARD_JSON = 10

STANDARD = 100

TRUFFLE = 2

VYPER = 7

WAFFLE = 8

crytic_compile.platform.vyper module

Vyper platform

class crytic_compile.platform.vyper.**Vyper**(*target: str*, ***kwargs*)
Bases: *crytic_compile.platform.abstract_platform.AbstractPlatform*

Vyper platform

NAME = 'vyper'

PROJECT_URL = 'https://github.com/vyperlang/vyper'

TYPE = 7

compile(*crytic_compile: CryticCompile*, ***kwargs*)
Compile the target

Parameters

- **crytic_compile** –
- **target** –
- **kwargs** –

Returns

is_dependency(*_path*)
Always return false

Parameters **_path** –

Returns

static is_supported(*target: str*, ***kwargs*) → bool
Check if the target is a vyper project

Parameters **target** –

Returns

crytic_compile.platform.waffle module

Waffle platform

class crytic_compile.platform.waffle.**Waffle**(*target: str*, ***kwargs*)
Bases: *crytic_compile.platform.abstract_platform.AbstractPlatform*

Waffle platform

NAME = 'Waffle'

PROJECT_URL = 'https://github.com/EthWorks/Waffle'

TYPE = 8

compile(*crytic_compile: CryticCompile*, ***kwargs*)
Compile the target

Parameters

- **crytic_compile** –
- **target** –
- **kwargs** –

Returns

is_dependency(*path: str*) → bool
Check if the path is a dependency

Parameters **path** –

Returns

static is_supported(*target: str*, ***kwargs*) → bool
Check if the target is a waffle project

Parameters **target** –

Returns

Module contents

Init module

1.1.1.4 crytic_compile.utils package

Submodules

crytic_compile.utils.naming module

Module handling the file naming operation (relative -> absolute, etc)

class crytic_compile.utils.naming.**Filename**(*absolute*, *used*, *relative*, *short*)
Bases: tuple

absolute
Alias for field number 0

relative
Alias for field number 2

short
Alias for field number 3

used
Alias for field number 1

crytic_compile.utils.naming.**combine_filename_name**(*filename: str*, *name: str*)
Combine the filename with the contract name

Parameters

- **filename** –
- **name** –

Returns

crytic_compile.utils.naming.**convert_filename**(*used_filename: Union[str, pathlib.Path]*, *relative_to_short*, *crytic_compile: CryticCompile*, *working_dir=None*) → crytic_compile.utils.naming.Filename
Convert filename. The used_filename can be absolute, relative, or missing node_modules/contracts directory convert_filename return a tuple(absolute,used), where absolute points to the absolute path, and used the original

Parameters

- **used_filename** –
- **relative_to_short** – lambda function
- **crytic_compile** –
- **working_dir** –

Returns Filename (namedtuple)

crytic_compile.utils.naming.**extract_filename**(*name: str*)
Convert '/path:Contract' to /path

crytic_compile.utils.naming.**extract_name**(*name: str*)
Convert '/path:Contract' to Contract

crytic_compile.utils.natspec module

Natspec module https://solidity.readthedocs.io/en/latest/natspec-format.html

class crytic_compile.utils.natspec.**DevDoc**(*devdoc: Dict[KT, VT]*)
Bases: object

Model the dev doc

author
Return the dev author

Returns Optional[str]

details
Return the dev details

Returns Optional[str]

export() → Dict[KT, VT]
Export to a python dict

Returns Dict

methods
Return the dev methods

Returns Dict[str, DevMethod]

title
Return the dev title

Returns Optional[str]

class crytic_compile.utils.natspec.**DevMethod**(*method*)
Bases: object

Model the dev method

author
Return the method author

Returns Optional[str]

details
Return the method details

Returns Optional[str]

export() → Dict[KT, VT]
Export to a python dict

Returns Dict

method_return
Return the method return

Returns Optional[str]

params
Return the method params

Returns Dict[str, str]

class crytic_compile.utils.natspec.**Natspec**(*userdoc: Dict[KT, VT], devdoc: Dict[KT, VT]*)
Bases: object

Model natspec

devdoc
Return the devdoc

Returns DevDoc

userdoc
Return the userdoc

Returns UserDoc

class crytic_compile.utils.natspec.**UserDoc**(*userdoc: dict*)
Bases: object

Model the user doc

export() → Dict[KT, VT]
Export to a python dict

Returns Dict

methods
Return the user methods

Returns Dict[str, UserMethod]

notice
Return the user notice

Returns Optional[str]

class crytic_compile.utils.natspec.**UserMethod**(*method*)
Bases: object

Model the user method

export() → Dict[KT, VT]
Export to a python dict

Returns Dict

notice
Return the method notice

Returns Optional[str]

crytic_compile.utils.npm module

Module handling NPM related features

crytic_compile.utils.npm.**get_package_name**(*target_txt: Union[str, SolcStandardJson]*) → Optional[str]
Return the package's name

Parameters **target_txt** –

Returns str or None

crytic_compile.utils.unit_tests module

Module handling unit-tests features

crytic_compile.utils.unit_tests.**guess_tests**(*target: str*) → List[str]
Try to guess the unit tests

Parameters **target** –

Returns

crytic_compile.utils.zip module

Handle ZIP operations

crytic_compile.utils.zip.**load_from_zip**(*target: str*) → List[crytic_compile.crytic_compile.CryticCompile]
Load a file from a zip

Parameters **target** –

Returns

crytic_compile.utils.zip.**save_to_zip**(*crytic_compiles: List[CryticCompile], zipfile: str*)
Save projects to a zip

Parameters

- **crytic_compiles** –
- **zipfile** –

Returns

Module contents

1.1.2 Submodules

1.1.3 crytic_compile.crytic_compile module

CryticCompile main module. Handle the compilation.

class crytic_compile.crytic_compile.**CryticCompile**(*target: Union[str, crytic_compile.platform.abstract_platform.AbstractPlatform], **kwargs*)

Bases: object

Main class.

abi(*name: str*) → Dict[KT, VT]
Get the ABI from a contract

Parameters **name** –

Returns

abis
Return the ABIs

Returns

absolute_filename_of_contract(*name: str*) → str

Returns Absolute filename

ast(*path: str*) → Optional[Dict[KT, VT]]
Return of the file

Parameters **path** –

Returns

asts
Return the ASTs

Returns dict (absolute filename -> AST)

bytecode_init(*name: str*, *libraries: Union[None, Dict[str, str]] = None*) → str
Return the init bytecode of the contract. If library is provided, patch the bytecode

Parameters

- **name** –
- **libraries** –

Returns

bytecode_only
Return true if only the bytecode was retrieved

Returns

bytecode_runtime (*name: str*, *libraries: Union[None*, *Dict[str*, *str]] = None*) → str
Return the runtime bytecode of the contract. If library is provided, patch the bytecode

Parameters

- **name** –
- **libraries** –

Returns

bytecodes_init
Return the init bytecodes

Returns

bytecodes_runtime
Return the runtime bytecodes

Returns

compiler_version
Return the compiler used as a namedtuple(compiler, version)

Returns

contracts_absolute_filenames
Return a dict (contract_name -> absolute filename)

Returns

contracts_filenames
Return a dict contract_name -> Filename namedtuple (absolute, used)

Returns dict(name -> utils.namings.Filename)

contracts_names
Return the contracts names

Returns

contracts_names_without_libraries
Return the contracts names (without the librairies)

Returns

dependencies
Return the dependencies files

Returns

export (***kwargs*) → Optional[str]
Export to json. The json format can be crytic-compile, solc or truffle.

filename_lookup (*filename: str*) → crytic_compile.utils.naming.Filename
Return a crytic_compile.naming.Filename from a any filename form (used/absolute/relative)

Parameters **filename** – str

Returns crytic_compile.naming.Filename

filename_of_contract (*name: str*) → crytic_compile.utils.naming.Filename

Returns utils.namings.Filename

filenames

Returns set(naming.Filename)

find_absolute_filename_from_used_filename (*used_filename: str*) → str
Return the absolute filename based on the used one

Parameters **used_filename** –

Returns absolute filename

hashes (*name: str*) → Dict[str, int]
Return the hashes of the functions

Parameters **name** –

Returns

static import_archive_compilations (*compiled_archive: Union[str, Dict[KT, VT]]*) → List[crytic_compile.crytic_compile.CryticCompile]
Import from an archive. compiled_archive is either a json file or the loaded dictionary The dictionary myst contain the "compilations" keyword

Parameters **compiled_archive** –

Returns

is_dependency (*filename: str*) → bool
Check if the filename is a dependency

Parameters **filename** –

Returns

libraries
Return the libraries used (contract_name -> [(library, pattern))])

Returns

libraries_names (*name: str*) → List[str]
Return the name of the libraries used by the contract

Parameters **name** – contract

Returns list of libraries name

libraries_names_and_patterns (*name*)
Return the name of the libraries used by the contract

Parameters **name** – contract

Returns list of (libraries name, pattern)

natspec
Return the natspec of the contractse

Returns Dict[str, Natspec]

package
Return the package name

Returns

package_name

Returns str or None

platform
Return the platform module

Returns

relative_filename_from_absolute_filename(*absolute_filename: str*) → str
Return the relative file based on the absolute name

Parameters **absolute_filename** –

Returns

src_content
Return the source content, filename -> source_code

Returns

src_content_for_file(*filename_absolute: str*) → Optional[str]
Get the source code of the file

Parameters **filename_absolute** –

Returns

srcmap_init(*name: str*) → List[str]
Return the init srcmap

Parameters **name** –

Returns

srcmap_runtime(*name: str*) → List[str]
Return the runtime srcmap

Parameters **name** –

Returns

srcmaps_init
Return the init srcmap

Returns

srcmaps_runtime
Return the runtime srcmap

Returns

target
Return the target (project)

Returns

type
Return the type of the platform used

Returns

used_filename_of_contract(*name: str*) → str

Returns Used filename

working_dir
Return the working dir

Returns

crytic_compile.crytic_compile.**compile_all**(*target: str*, ***kwargs*) → List[crytic_compile.crytic_compile.CryticCompile]

Given a direct or glob pattern target, compiles all underlying sources and returns all the relevant instances of CryticCompile.

Parameters

- **target** – A string representing a file/directory path or glob pattern denoting where compilation shouldoccur.
- **kwargs** – The remainder of the arguments passed through to all compilation steps.

Returns Returns a list of CryticCompile instances for all compilations which occurred.

crytic_compile.crytic_compile.**get_platforms**() → List[Type[crytic_compile.platform.abstract_platform.AbstractPla

Return the available platforms classes

Returns

crytic_compile.crytic_compile.**is_supported**(*target: str*) → bool

Check if the target is supported

Parameters **target** –

Returns

1.1.4 Module contents

Init module

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W