py-solc-x Documentation

Release 1.1.1

Ben Hauser

Oct 12, 2021

Contents

CHAPTER 1

py-solc-x

Python wrapper and version management tool for the `solc` Solidity compiler.

1.1 Features

- Full support for Solidity versions `>=0.4.11`
- Installs Solidity on Linux, OSX and Windows
- Compiles Solidity from source on Linux and OSX

1.2 Credit

py-solc-x is forked from py-solc which was written by Piper Merriam.

1.3 Dependencies

Py-solc-x allows the use of multiple versions of solc, and can install or compile them as needed. If you wish to compile from source you must first install the required solc dependencies.

CHAPTER 2

Using the Compiler

py-solc-x provides several functions that you can use to interact with the `solc` compiler.

2.1 Compiling a Source String

`solcx.`**`compile_source`**(*source*, ***kwargs*)

Compile a Solidity contract.

Compilation is handled via the `--combined-json` flag. Depending on the Solidity version used, some keyword arguments may not be available.

Returns a dict, where each top-level key is a contract. The filename will be `<stdin>`.

```
>>> import solcx
>>> solcx.compile_source(
...     "contract Foo { function bar() public { return; } }",
...     output_values=["abi", "bin-runtime"],
...     solc_version="0.7.0"
... )
{
    '<stdin>:Foo': {
        'abi': [{'inputs': [], 'name': 'bar', 'outputs': [], 'stateMutability':
↪'nonpayable', 'type': 'function'}],
        'bin-runtime':
↪'60806040523480156005760003560e01c8063febb0f7e14602d575b600080fd5b6
↪'
    }
}
```

Required Arguments

`source` **str** Solidity contract to be compiled.

Optional py-solc-x Arguments

solc_binary str | Path Path of the `solc` binary to use. May be given as a string or `Path` object. If not given, the currently active version is used (as set by *solcx.set_solc_version*)

solc_version str | Version `solc` version to use. May be given as a string or `Version` object. If not given, the currently active version is used. Ignored if `solc_binary` is also given.

allow_empty bool If `True`, do not raise when no compiled contracts are returned. Defaults to `False`.

Optional Compiler Arguments

Depending on the Solidity version used, using some of these arguments may raise `UnknownOption`. See the documentation for your target Solidity version for more information.

output_values List Compiler outputs to return. Valid options depend on the version of `solc`. If not given, all possible outputs for the active version are returned.

import_remappings Dict | List | str Path remappings. May be given as a string or list of strings formatted as `"prefix=path"`, or a dict of `{"prefix": "path"}`.

base_path Path | str Use the given path as the root of the source tree instead of the root of the filesystem.

allow_paths List | Path | str A path, or list of paths, to allow for imports.

output_dir str Creates one file per component and contract/file at the specified directory.

overwrite bool Overwrite existing files (used in combination with `output_dir`)

evm_version str Select the desired EVM version. Valid options depend on the `solc` version.

revert_strings List | str Strip revert (and require) reason strings or add additional debugging information.

metadata_hash str Choose hash method for the bytecode metadata or disable it.

metadata_literal bool Store referenced sources as literal data in the metadata output.

optimize bool Enable bytecode optimizer.

optimize_runs int Set for how many contract runs to optimize. Lower values will optimize more for initial deployment cost, higher values will optimize more for high-frequency usage.

optimize_yul bool Enable the yul optimizer.

no_optimize_yul bool Disable the yul optimizer.

yul_optimizations int Force yul optimizer to use the specified sequence of optimization steps instead of the built-in one.

2.2 Compiling Files

`solcx.`**compile_files**(*source*, ***kwargs*)

Compile one or more Solidity source files.

Compilation is handled via the `--combined-json` flag. Depending on the Solidity version used, some keyword arguments may not be available.

Returns a dict, where each top-level key is a contract.

```
>>> import solcx
>>> solcx.compile_files(
...     ["Foo.sol"],
...     output_values=["abi", "bin-runtime"],
...     solc_version="0.7.0"
... )
{
    '<stdin>:Foo': {
        'abi': [{'inputs': [], 'name': 'bar', 'outputs': [], 'stateMutability':
↪'nonpayable', 'type': 'function'}],
        'bin-runtime':
↪'60806040523480156000f57600080fd5b506004361060285760003560e01c8063febb0f7e14602d575b600080fd5b6
↪'
    }
}
```

Required Arguments

source_files List | Path | str Solidity source file, or list of source files, to be compiled. Files may be given as strings or `Path` objects.

Optional py-solc-x Arguments

solc_binary str | Path Path of the `solc` binary to use. May be given as a string or `Path` object. If not given, the currently active version is used (as set by *solcx.set_solc_version*)

solc_version str | Version `solc` version to use. May be given as a string or `Version` object. If not given, the currently active version is used. Ignored if `solc_binary` is also given.

allow_empty bool If `True`, do not raise when no compiled contracts are returned. Defaults to `False`.

Optional Compiler Arguments

Depending on the Solidity version used, using some of these arguments may raise `UnknownOption`. See the documentation for your target Solidity version for more information.

output_values List Compiler outputs to return. Valid options depend on the version of `solc`. If not given, all possible outputs for the active version are returned.

import_remappings Dict | List | str Path remappings. May be given as a string or list of strings formatted as `"prefix=path"`, or a dict of `{"prefix": "path"}`.

base_path Path | str Use the given path as the root of the source tree instead of the root of the filesystem.

allow_paths List | Path | str A path, or list of paths, to allow for imports.

output_dir str Creates one file per component and contract/file at the specified directory.

overwrite bool Overwrite existing files (used in combination with `output_dir`)

evm_version str Select the desired EVM version. Valid options depend on the `solc` version.

revert_strings List | str Strip revert (and require) reason strings or add additional debugging information.

metadata_hash str Choose hash method for the bytecode metadata or disable it.

metadata_literal bool Store referenced sources as literal data in the metadata output.

optimize bool Enable bytecode optimizer.

optimize_runs int Set for how many contract runs to optimize. Lower values will optimize more for initial deployment cost, higher values will optimize more for high-frequency usage.

optimize_yul bool Enable the yul optimizer.

no_optimize_yul bool Disable the yul optimizer.

yul_optimizations int Force yul optimizer to use the specified sequence of optimization steps instead of the built-in one.

2.3 Compiling with the Standard JSON Format

`solcx.`**`compile_standard`**(*input_data*, ***kwargs*)

Compile Solidity contracts using the JSON-input-output interface.

See the Solidity documentation on the compiler input-output JSON for details on the expected JSON input and output formats.

Required Arguments

input_data Dict Compiler JSON input.

Optional py-solc-x Arguments

solc_binary str | Path Path of the `solc` binary to use. May be given as a string or `Path` object. If not given, the currently active version is used (as set by *`solcx.set_solc_version`*)

solc_version str | Version `solc` version to use. May be given as a string or `Version` object. If not given, the currently active version is used. Ignored if `solc_binary` is also given.

allow_empty bool If `True`, do not raise when no compiled contracts are returned. Defaults to `False`.

Optional Compiler Arguments

Depending on the Solidity version used, using some of these arguments may raise `UnknownOption`. See the documentation for your target Solidity version for more information.

base_path Path | str Use the given path as the root of the source tree instead of the root of the filesystem.

allow_paths List | Path | str A path, or list of paths, to allow for imports.

output_dir str Creates one file per component and contract/file at the specified directory.

overwrite bool Overwrite existing files (used in combination with `output_dir`)

2.4 Linking Libraries

`solcx.`**`link_code`**(*unlinked_bytecode*, *libraries*, *solc_binary=None*, *solc_version=None*)

Add library addresses into unlinked bytecode.

See the Solidity documentation on using the commandline compiler for more information on linking libraries.

Returns the linked bytecode as a string.

```
>>> import solcx
>>> unlinked_bytecode =
↪"6060604052607680601060003960000f3606060405260e060020a6000350463e7f09e058114601a575b005b60187f0
↪_TestA_____________________________________
↪90630c55699c906064906000906004818660325a03f41560025750505056"

>>> solcx.link_code(
...     unlinked_bytecode,
...     {'TestA': "0xd3cda913deb6f67967b99d67acdfa1712c293601"}
... )

↪"6060604052607680601060003960000f3606060405260e060020a6000350463e7f09e058114601a575b005b60187f0
↪"
```

Required Arguments

unlinked_bytecode str Compiled bytecode containing one or more library placeholders.

libraries Dict Library addresses given as `{"library name": "address"}`

Optional py-solc-x Arguments

solc_binary str | Path Path of the `solc` binary to use. May be given as a string or `Path` object. If not given, the currently active version is used (as set by *solcx.set_solc_version*)

solc_version str | Version `solc` version to use. May be given as a string or `Version` object. If not given, the currently active version is used. Ignored if `solc_binary` is also given.

CHAPTER 3

The Low-Level Process Wrapper

Along with the *main compiler functions*, you can also directly call `solc` using the low-level wrapper.

solc_wrapper(*solc_binary=None*, *stdin=None*, *source_files=None*, *import_remappings=None*, *success_return_code=None*, ***kwargs*)

Wrapper function for calling to `solc`.

Returns the process `stdout` as a string, `stderr` as a string, the full command executed as a list of strings, and the completed `Popen` object used to call `solc`.

Arguments

solc_binary [Path | str] Location of the `solc` binary. If not given, the current default binary is used.

stdin [str] Input to pass to `solc` via stdin

source_files **List | Path | str** Solidity source file, or list of source files, to be compiled. Files may be given as strings or `Path` objects.

import_remappings [Dict | List | str] Path remappings. May be given as a string or list of strings formatted as `"prefix=path"` or a dict of `{"prefix": "path"}`

success_return_code [int] Expected exit code. Raises `SolcError` if the process returns a different value. Defaults to `0`.

****kwargs** **Any** Flags to be passed to *solc*. Keywords are converted to flags by prepending `--` and replacing `_` with `-`, for example the keyword `evm_version` becomes `--evm-version`. Values may be given in the following formats:

- `False` or `None`: The flag is ignored
- `True`: The flag is passed to the compiler without any arguments
- `str`: The value is given as an argument without any modification
- `int`: The value is converted to a string
- `Path`: The value is converted to a string via `Path.as_posix`
- `List` or `Tuple`: Elements in the sequence are converted to strings and joined with `,`

CHAPTER 4

Solidity Version Management

4.1 Installation Folder

By default, `solc` versions are installed at `~/.solcx/`. Each installed version is named using the following pattern: `solc-v[MAJOR].[MINOR].[PATH]`

If you wish to install to a different directory you can specify it with the `SOLCX_BINARY_PATH` environment variable. You can also give a custom directory to most installation functions using the optional `solcx_binary_path` keyword argument.

`solcx.`**`get_solcx_install_folder`**(*solcx_binary_path=None*)

Return the directory where py-solc-x stores installed `solc` binaries.

```
>>> solcx.get_solcx_install_folder()
PosixPath('/home/computer/.solcx')
```

4.2 Getting and Setting the Active Version

When py-solc-x is imported, it attempts to locate an installed version of `solc` using `which` on Linux or OSX systems, or `where.exe` on Windows. If found, this version is set as the active version. If not found, it uses the latest version that has been installed by py-solc-x.

4.2.1 Getting the Active Version

Use the following methods to check the active `solc` version:

`solcx.`**`get_solc_version`**(*with_commit_hash=False*)

Return the version of the current active `solc` binary, as a `Version` object.

- `with_commit_hash`: If `True`, the returned version includes the commit hash

```
>>> solcx.get_solc_version()
Version('0.7.0')

>>> solcx.get_solc_version(True)
Version('0.7.0+commit.9e61f92b')
```

solcx.install.**get_executable**(*version=None*, *solcx_binary_path=None*)

Return a `Path` object for a `solc` binary.

If no arguments are given, returns the current active version. If a version is specified, returns the installed binary matching the given version.

Raises `SolcNotInstalled` if no binary is found.

```
>>> solcx.install.get_executable()
PosixPath('/usr/bin/solc')
```

solcx.**get_installed_solc_versions**(*solcx_binary_path=None*)

Return a list of currently installed `solc` versions.

```
>>> solcx.get_installed_solc_versions()
[Version('0.7.0'), Version('0.6.8'), Version('0.6.3'), Version('0.5.7'), Version(
↪'0.4.25')]
```

4.2.2 Setting the Active Version

solcx.**set_solc_version**(*version*, *silent=False*, *solcx_binary_path=None*)

Set the currently active `solc` version.

```
>>> solcx.set_solc_version('0.5.0')
```

solcx.**set_solc_version_pragma**(*pragma_string*, *silent=False*, *check_new=False*)

Set the currently active `solc` binary based on a pragma statement.

The newest installed version that matches the pragma is chosen. Raises `SolcNotInstalled` if no installed versions match.

```
>>> solcx.set_solc_version_pragma('pragma solidity ^0.5.0;')
Version('0.5.17')
```

4.3 Importing Already-Installed Versions

solcx.**import_installed_solc**(*solcx_binary_path=None*)

Search for and copy installed `solc` versions into the local installation folder.

This function is especially useful on OSX, to access Solidity versions that you have installed from homebrew and where a precompiled binary is not available.

```
>>> solcx.import_installed_solc()
[Version('0.7.0'), Version('0.6.12')]
```

4.4 Installing Solidity

py-solc-x downloads and installs precompiled binaries from solc-bin.ethereum.org. Different binaries are available depending on your operating system.

4.4.1 Getting Installable Versions

`solcx.get_installable_solc_versions()`
Return a list of all `solc` versions that can be installed by py-solc-x.

```
>>> solcx.get_installable_solc_versions()
[Version('0.7.0'), Version('0.6.12'), Version('0.6.11'), Version('0.6.10'),␣
↪Version('0.6.9'), Version('0.6.8'), Version('0.6.7'), Version('0.6.6'), Version(
↪'0.6.5'), Version('0.6.4'), Version('0.6.3'), Version('0.6.2'), Version('0.6.1
↪'), Version('0.6.0'), Version('0.5.17'), Version('0.5.16'), Version('0.5.15'),␣
↪Version('0.5.14'), Version('0.5.13'), Version('0.5.12'), Version('0.5.11'),␣
↪Version('0.5.10'), Version('0.5.9'), Version('0.5.8'), Version('0.5.7'),␣
↪Version('0.5.6'), Version('0.5.5'), Version('0.5.4'), Version('0.5.3'), Version(
↪'0.5.2'), Version('0.5.1'), Version('0.5.0'), Version('0.4.26'), Version('0.4.25
↪'), Version('0.4.24'), Version('0.4.23'), Version('0.4.22'), Version('0.4.21'),␣
↪Version('0.4.20'), Version('0.4.19'), Version('0.4.18'), Version('0.4.17'),␣
↪Version('0.4.16'), Version('0.4.15'), Version('0.4.14'), Version('0.4.13'),␣
↪Version('0.4.12'), Version('0.4.11')]
```

4.4.2 Installing Precompiled Binaries

`solcx.install_solc(version="latest", show_progress=False, solcx_binary_path=None)`
Download and install a precompiled `solc` binary.

`version` str | Version Version of `solc` to install. Default is the newest available version.

`show_progress` bool If `True`, display a progress bar while downloading. Requires installing the tqdm package.

`solcx_binary_path` Path | str User-defined path, used to override the default installation directory.

4.5 Building from Source

When a precompiled version of Solidity isn't available for your operating system, you may still install it by building from the source code. Source code is downloaded from Github.

Note: If you wish to compile from source you must first install the required solc dependencies.

4.5.1 Getting Compilable Versions

`solcx.get_compilable_solc_versions(headers=None)`
Return a list of all `solc` versions that can be installed by py-solc-x.

`headers` Dict Headers to include in the request to Github.

```
>>> solcx.get_compilable_solc_versions()
[Version('0.7.0'), Version('0.6.12'), Version('0.6.11'), Version('0.6.10'),␣
↪Version('0.6.9'), Version('0.6.8'), Version('0.6.7'), Version('0.6.6'), Version(
↪'0.6.5'), Version('0.6.4'), Version('0.6.3'), Version('0.6.2'), Version('0.6.1
↪'), Version('0.6.0'), Version('0.5.17'), Version('0.5.16'), Version('0.5.15'),␣
↪Version('0.5.14'), Version('0.5.13'), Version('0.5.12'), Version('0.5.11'),␣
↪Version('0.5.10'), Version('0.5.9'), Version('0.5.8'), Version('0.5.7'),␣
↪Version('0.5.6'), Version('0.5.5'), Version('0.5.4'), Version('0.5.3'), Version(
↪'0.5.2'), Version('0.5.1'), Version('0.5.0'), Version('0.4.26'), Version('0.4.25
↪'), Version('0.4.24'), Version('0.4.23'), Version('0.4.22'), Version('0.4.21'),␣
↪Version('0.4.20'), Version('0.4.19'), Version('0.4.18'), Version('0.4.17'),␣
↪Version('0.4.16'), Version('0.4.15'), Version('0.4.14'), Version('0.4.13'),␣
↪Version('0.4.12'), Version('0.4.11')]
```

4.5.2 Compiling Solidity from Source

`solcx.`**`compile_solc`**(*version*, *show_progress=False*, *solcx_binary_path=None*)

Install a version of `solc` by downloading and compiling source code.

This function is only available when using Linux or OSX.

Arguments:

`version` **str | Version** Version of `solc` to install.

`show_progress` **bool** If `True`, display a progress bar while downloading. Requires installing the tqdm package.

`solcx_binary_path` **Path | str** User-defined path, used to override the default installation directory.

Index

S