# User

I have cloned a repository from github which builds rdrand python package. I observed that it does not include setup.cfg file but only following files:

Mode LastWriteTime Length Name

---- ------------- ------ ----

d---- 12-02-2024 15:08 docs
-a--- 12-02-2024 15:08 1190 .gitignore
-a--- 12-02-2024 15:08 62 .readthedocs.yml
-a--- 12-02-2024 15:08 1466 LICENSE
-a--- 12-02-2024 15:08 14104 rdrand.c
-a--- 12-02-2024 15:08 3440 rdrand.py
-a--- 12-02-2024 15:08 1893 rdrandom.rst
-a--- 12-02-2024 15:08 2235 README.md
-a--- 12-02-2024 15:08 1730 README.txt
-a--- 12-02-2024 15:08 1163 setup.py

Is there a way to create setup.cfg and use that to build the package?

# Bing

Yes, you can create a setup.cfg file to specify configuration options for setuptools. Here's a basic example of what the setup.cfg file could look like for your project:

[metadata]

name = rdrand

version = 0.1.0

description = A Python package for generating random numbers using rdrand

long\_description = file: README.md

long\_description\_content\_type = text/markdown

author = Your Name

author\_email = your.email@example.com

url = https://github.com/yourusername/rdrand

license = MIT

[options]

packages = find:

python\_requires = >=3.6

This is a very basic example and you might need to adjust it according to your project's needs. For instance, if your project has any dependencies, you can specify them under [options] like so:

[options]

install\_requires =

numpy

pandas

Once you have your setup.cfg, you can build your package using the following command:

python setup.py sdist bdist\_wheel

This will create a source distribution (.tar.gz) and a wheel distribution (.whl) in a newly created dist/ directory. You can then install your package using pip:

pip install dist/\*.whl

Remember to replace the placeholders in the setup.cfg with your actual project information. Also, please note that the setup.cfg file should be in the same directory as your setup.py file. Hope this helps! Let me know if you have any other questions. 😊

# User

I have another setup.cfg from another package fastparquet as under: Can you modify your above example configuration in line with that:

[aliases]
test=pytest

[tool:pytest]
addopts = -vv

# See the docstring in versioneer.py for instructions. Note that you must

# re-run 'versioneer.py setup' after changing this section, and commit the

# resulting files.

[versioneer]
VCS = git
style = pep440
versionfile\_source = fastparquet/\_version.py
versionfile\_build = fastparquet/\_version.py
tag\_prefix = ""

# Bing

Sure, I can help you modify the setup.cfg example to align with the structure of the fastparquet package's setup.cfg. Here's how it might look:

[metadata]

name = rdrand

version = 0.1.0

description = A Python package for generating random numbers using rdrand

long\_description = file: README.md

long\_description\_content\_type = text/markdown

author = Your Name

author\_email = your.email@example.com

url = https://github.com/yourusername/rdrand

license = MIT

[options]

packages = find:

python\_requires = >=3.6

[aliases]

test=pytest

[tool:pytest]

addopts = -vv

[versioneer]

VCS = git

style = pep440

versionfile\_source = rdrand/\_version.py

versionfile\_build = rdrand/\_version.py

tag\_prefix = ""

This configuration includes the pytest and versioneer sections from the fastparquet package's setup.cfg. You'll need to create a \_version.py file in your rdrand directory for versioneer to work. Also, remember to replace the placeholders with your actual project information.

After creating your setup.cfg, you can build your package using the following command:

python setup.py sdist bdist\_wheel

This will create a source distribution (.tar.gz) and a wheel distribution (.whl) in a newly created dist/ directory. You can then install your package using pip:

pip install dist/\*.whl

Let me know if you have any other questions! 😊

# User

What is the latest pep in place of pep440 which I need to replace? I have Python 3.12.2.

# Bing

PEP 440 is still the current standard for Python package versions[**1**](https://peps.python.org/pep-0440/)[**3**](https://docs.python.org/3/whatsnew/3.12.html). It has not been superseded by a newer PEP as of my knowledge cutoff in 2021. However, I recommend checking the official Python Enhancement Proposals (PEPs) page or other reliable sources to get the most up-to-date information. Please note that the versioning scheme defined in PEP 440 is compatible with Python 3.12.2[**4**](https://peps.python.org/pep-0619/). If there are any changes, they would be documented in the official Python documentation or in a new PEP. Let me know if you have any other questions! 😊

Source(s)

1. [PEP 440 – Version Identification and Dependency Specification - Python](https://peps.python.org/pep-0440/)

2. [Quirks of Python package versioning](https://sethmlarson.dev/pep-440)

3. [What’s New In Python 3.12 — Python 3.12.2 documentation](https://docs.python.org/3/whatsnew/3.12.html)

4. [PEP 619 – Python 3.10 Release Schedule | peps.python.org](https://peps.python.org/pep-0619/)

5. [PEP 602 – Annual Release Cycle for Python | peps.python.org](https://peps.python.org/pep-0602/)

6. <https://github.com/python/peps/blob/main/peps/pep-0619.rst>

# User

Please include following details in metadata in place of what you have done above:

Name: rdrand
Version: 1.5.0
Summary: Python interface to Intel hardware rng
Home-page: https://github.com/stillson/rdrand
Author: Chris Stillson
Author-email: stillson@gmail.com
License: New BSD license
Location: D:\Programs\Python\Lib\site-packages

# Bing

Sure, here's how your setup.cfg file would look with the provided metadata:

[metadata]

name = rdrand

version = 1.5.0

description = Python interface to Intel hardware rng

long\_description = file: README.md

long\_description\_content\_type = text/markdown

author = Chris Stillson

author\_email = stillson@gmail.com

url = https://github.com/stillson/rdrand

license = New BSD license

[options]

packages = find:

python\_requires = >=3.6

[aliases]

test=pytest

[tool:pytest]

addopts = -vv

[versioneer]

VCS = git

style = pep440

versionfile\_source = rdrand/\_version.py

versionfile\_build = rdrand/\_version.py

tag\_prefix = ""

Please note that the Location field is not typically included in the setup.cfg file. The location of the package is determined by the system when you install it, and it can vary depending on the system and the specific Python environment you're using.

You can build your package using the following command:

python setup.py sdist bdist\_wheel

This will create a source distribution (.tar.gz) and a wheel distribution (.whl) in a newly created dist/ directory. You can then install your package using pip:

pip install dist/\*.whl

Let me know if you have any other questions! 😊

# User

See errors below when tried to build:

PS D:\Downloads\rdrand\_12022024> pip install .
Processing d:\downloads\rdrand\_12022024
Preparing metadata (setup.py) ... done
Building wheels for collected packages: rdrand
Building wheel for rdrand (setup.py) ... error
error: subprocess-exited-with-error

× python setup.py bdist\_wheel did not run successfully.
│ exit code: 1
╰─> [6 lines of output]
gcc: warning: argument unused during compilation: '-mdll' [-Wunused-command-line-argument]
rdrand.c:378:9: error: call to undeclared function 'bcopy'; ISO C99 and later do not support implicit function declarations [-Wimplicit-function-declaration]
378 | bcopy((char\*)&rando, &data[num\_quads \* 8], num\_chars);
| ^
1 error generated.
error: command 'D:\Programs\Msys2\clang64\bin\gcc.exe' failed with exit code 1
[end of output]

note: This error originates from a subprocess, and is likely not a problem with pip.
ERROR: Failed building wheel for rdrand
Running setup.py clean for rdrand
Failed to build rdrand
ERROR: Could not build wheels for rdrand, which is required to install pyproject.toml-based projects

PS D:\Downloads\rdrand\_12022024> pip install dist/*.whl
WARNING: Requirement 'dist/*.whl' looks like a filename, but the file does not exist
ERROR: \*.whl is not a valid wheel filename.

I tried both pip install . and pip install dist/\*.whl - both failed as you can see above.

# Bing

The error message indicates that the C compiler is unable to find the bcopy function. This function is a legacy function in Unix-like systems, and it's not part of the standard C library. In modern code, it's recommended to use memcpy instead of bcopy.

You can fix this issue by replacing the bcopy call with memcpy. The bcopy function is used like this: bcopy(src, dest, n), and the equivalent memcpy call would be memcpy(dest, src, n).

So, you should replace the line in rdrand.c:

bcopy((char\*)&rando, &data[num\_quads \* 8], num\_chars);

with:

memcpy(&data[num\_quads \* 8], (char\*)&rando, num\_chars);

After making this change, try building the package again with pip install ..

As for the pip install dist/\*.whl command, it failed because the previous command did not successfully build the wheel file due to the error. Once you fix the error and successfully build the package, you should be able to install the wheel file with pip install dist/\*.whl.

Let me know if you have any other questions or if you encounter any other issues! 😊

# User

Errors after above change:

Building wheels for collected packages: rdrand
Building wheel for rdrand (setup.py) ... error
error: subprocess-exited-with-error

× python setup.py bdist\_wheel did not run successfully.
│ exit code: 1
╰─> [111 lines of output]
gcc: warning: argument unused during compilation: '-mdll' [-Wunused-command-line-argument]
rdrand.c:104:18: error: instruction requires: Not 64-bit mode
104 | asm volatile("pushl %%ebx \n\t" /\* save %ebx */
| ^
<inline asm>:1:2: note: instantiated into assembly here
1 | pushl %ebx
| ^
rdrand.c:106:38: error: instruction requires: Not 64-bit mode
106 | "movl %%ebx, %1 \n\t" /* save what cpuid just put in %ebx */
| ^
<inline asm>:4:2: note: instantiated into assembly here
4 | popl %ebx
| ^
rdrand.c:104:18: error: instruction requires: Not 64-bit mode
104 | asm volatile("pushl %%ebx \n\t" /* save %ebx */
| ^
<inline asm>:1:2: note: instantiated into assembly here
1 | pushl %ebx
| ^
rdrand.c:106:38: error: instruction requires: Not 64-bit mode
106 | "movl %%ebx, %1 \n\t" /* save what cpuid just put in %ebx */
| ^
<inline asm>:4:2: note: instantiated into assembly here
4 | popl %ebx
| ^
rdrand.c:104:18: error: instruction requires: Not 64-bit mode
104 | asm volatile("pushl %%ebx \n\t" /* save %ebx */
| ^
<inline asm>:1:2: note: instantiated into assembly here
1 | pushl %ebx
| ^
rdrand.c:106:38: error: instruction requires: Not 64-bit mode
106 | "movl %%ebx, %1 \n\t" /* save what cpuid just put in %ebx */
| ^
<inline asm>:4:2: note: instantiated into assembly here
4 | popl %ebx
| ^
rdrand.c:104:18: error: instruction requires: Not 64-bit mode
104 | asm volatile("pushl %%ebx \n\t" /* save %ebx */
| ^
<inline asm>:1:2: note: instantiated into assembly here
1 | pushl %ebx
| ^
rdrand.c:106:38: error: instruction requires: Not 64-bit mode
106 | "movl %%ebx, %1 \n\t" /* save what cpuid just put in %ebx */
| ^
<inline asm>:4:2: note: instantiated into assembly here
4 | popl %ebx
| ^
rdrand.c:104:18: error: instruction requires: Not 64-bit mode
104 | asm volatile("pushl %%ebx \n\t" /* save %ebx */
| ^
<inline asm>:1:2: note: instantiated into assembly here
1 | pushl %ebx
| ^
rdrand.c:106:38: error: instruction requires: Not 64-bit mode
106 | "movl %%ebx, %1 \n\t" /* save what cpuid just put in %ebx */
| ^
<inline asm>:4:2: note: instantiated into assembly here
4 | popl %ebx
| ^
rdrand.c:104:18: error: instruction requires: Not 64-bit mode
104 | asm volatile("pushl %%ebx \n\t" /* save %ebx */
| ^
<inline asm>:1:2: note: instantiated into assembly here
1 | pushl %ebx
| ^
rdrand.c:106:38: error: instruction requires: Not 64-bit mode
106 | "movl %%ebx, %1 \n\t" /* save what cpuid just put in %ebx \*/
| ^
<inli

# Bing

The error messages indicate that the assembly instructions in your rdrand.c file are not compatible with 64-bit mode. The pushl and popl instructions are intended for use in 32-bit mode, but it appears that your environment is compiling the code in 64-bit mode.

You can try replacing these instructions with their 64-bit counterparts, pushq and popq. However, please note that modifying assembly code can be tricky and may lead to unexpected behavior if not done correctly.

Here's how you might modify the assembly code:

asm volatile("pushq %%rbx \n\t" /\* save %ebx \*/

"cpuid \n\t"

"movl %%ebx, %1 \n\t" /\* save what cpuid just put in %ebx \*/

"popq %%rbx \n\t" /\* restore the old %ebx \*/

: "=a"(eax), "=r"(ebx), "=c"(ecx), "=d"(edx)

: "a"(functionnumber), "c"(0)

: "cc");

Please note that this is just a suggestion and might not work depending on the rest of your code. If you're not comfortable with assembly, I would recommend seeking help from someone who is, or consider using a higher-level method to achieve your goal. Let me know if you have any other questions! 😊

# User

See I have included:

[build]
compiler = mingw32

at the end of setup.cfg. That causes this error? Or do you I need to change c source code as above?

# Bing

The [build] compiler = mingw32 line in your setup.cfg file tells setuptools to use the MinGW compiler (which is a port of GCC for Windows) to build the C extension. This should not cause the error you're seeing, as the error is due to the assembly instructions in your C code, not the compiler itself.

The error messages indicate that the assembly instructions in your rdrand.c file are not compatible with 64-bit mode. The pushl and popl instructions are intended for use in 32-bit mode, but it appears that your environment is compiling the code in 64-bit mode.

You can try replacing these instructions with their 64-bit counterparts, pushq and popq. However, please note that modifying assembly code can be tricky and may lead to unexpected behavior if not done correctly.

Here's how you might modify the assembly code:

asm volatile("pushq %%rbx \n\t" /\* save %ebx \*/

"cpuid \n\t"

"movl %%ebx, %1 \n\t" /\* save what cpuid just put in %ebx \*/

"popq %%rbx \n\t" /\* restore the old %ebx \*/

: "=a"(eax), "=r"(ebx), "=c"(ecx), "=d"(edx)

: "a"(functionnumber), "c"(0)

: "cc");

Please note that this is just a suggestion and might not work depending on the rest of your code. If you're not comfortable with assembly, I would recommend seeking help from someone who is, or consider using a higher-level method to achieve your goal. Let me know if you have any other questions! 😊

# User

Some more related errors as below:

gcc: warning: argument unused during compilation: '-mdll' [-Wunused-command-line-argument]

rdrand.c:114:21: error: use of undeclared identifier 'eax'

114 | : "=a"(eax), "=r"(ebx), "=c"(ecx), "=d"(edx)

| ^

1 error generated.

error: command 'D:\\Programs\\Msys2\\clang64\\bin\\gcc.exe' failed with exit code 1

[end of output]

# Bing

The error message indicates that the variable eax is undeclared. This means that the compiler doesn't know what eax is because it hasn't been defined anywhere in your code.

In the context of your assembly code, eax, ebx, ecx, and edx are meant to represent registers. However, in C code, you need to declare these variables before you can use them.

Here's an example of how you might declare these variables:

int eax, ebx, ecx, edx;

You should add this line before the assembly code where these variables are used. After declaring these variables, the compiler should be able to recognize eax, ebx, ecx, and edx.

Please note that modifying assembly code can be tricky and may lead to unexpected behavior if not done correctly. If you're not comfortable with assembly, I would recommend seeking help from someone who is, or consider using a higher-level method to achieve your goal. Let me know if you have any other questions! 😊

# User

Should I define above in the beginning of file and if not where exactly?

# Bing

The variables eax, ebx, ecx, and edx should be declared in the function where the assembly code is located, not at the beginning of the file. They should be declared before they are used in the assembly code.

Here's an example of how you might do this:

void some\_function() {

int eax, ebx, ecx, edx;

asm volatile("pushq %%rbx \n\t" /\* save %ebx \*/

"cpuid \n\t"

"movl %%ebx, %1 \n\t" /\* save what cpuid just put in %ebx \*/

"popq %%rbx \n\t" /\* restore the old %ebx \*/

: "=a"(eax), "=r"(ebx), "=c"(ecx), "=d"(edx)

: "a"(functionnumber), "c"(0)

: "cc");

// rest of your code...

}

In this example, some\_function is the function where the assembly code is located. The variables eax, ebx, ecx, and edx are declared at the beginning of this function, before they are used in the assembly code.

Please replace some\_function and functionnumber with the actual function name and number in your code. Let me know if you have any other questions! 😊

# User

See my original assembly function and do all above changes recommended by you (all) in that:

// asm volatile("pushq %%ebx \n\t" /\* save %ebx
// "cpuid \n\t"
// "movl %%ebx, %1 \n\t" /\* save what cpuid just put in %ebx /// "popl %%ebx \n\t" /\* restore the old %ebx
// : "=a"(reg[0]), "=r"(reg[1]), "=c"(reg[2]), "=d"(reg[3])
// : "a"(op), "c"(subfunc)
// : "cc");

# Bing

Sure, here's how your assembly function might look after applying the changes:

int reg[4], op, subfunc;

asm volatile("pushq %%rbx \n\t" /\* save %ebx \*/

"cpuid \n\t"

"movl %%ebx, %1 \n\t" /\* save what cpuid just put in %ebx \*/

"popq %%rbx \n\t" /\* restore the old %ebx \*/

: "=a"(reg[0]), "=r"(reg[1]), "=c"(reg[2]), "=d"(reg[3])

: "a"(op), "c"(subfunc)

: "cc");

In this code:

* reg, op, and subfunc are declared before the assembly code. reg is an array of 4 integers, and op and subfunc are integers.
* The pushl and popl instructions have been replaced with pushq and popq, respectively, for compatibility with 64-bit mode.
* The bcopy function has been replaced with memcpy.

Please replace op and subfunc with the actual operation and sub-function numbers in your code. Let me know if you have any other questions! 😊

Source(s)

1. [github.com](https://github.com/intel/kernelflinger/tree/9a84515a0f2b6b5697c49c770513b8c432b6f9d4/libkernelflinger%2Flib.c)

# User

Errors:

error: subprocess-exited-with-error

× python setup.py bdist\_wheel did not run successfully.
│ exit code: 1
╰─> [21 lines of output]
gcc: warning: argument unused during compilation: '-mdll' [-Wunused-command-line-argument]
rdrand.c:110:5: error: redefinition of 'reg' with a different type: 'int[4]' vs 'unsigned int \*'
110 | int reg[4], op, subfunc;
| ^
rdrand.c:98:59: note: previous definition is here
98 | cpuid(unsigned int op, unsigned int subfunc, unsigned int reg[4])
| ^
rdrand.c:110:13: error: redefinition of 'op' with a different type: 'int' vs 'unsigned int'
110 | int reg[4], op, subfunc;
| ^
rdrand.c:98:20: note: previous definition is here
98 | cpuid(unsigned int op, unsigned int subfunc, unsigned int reg[4])
| ^
rdrand.c:110:17: error: redefinition of 'subfunc' with a different type: 'int' vs 'unsigned int'
110 | int reg[4], op, subfunc;
| ^
rdrand.c:98:37: note: previous definition is here
98 | cpuid(unsigned int op, unsigned int subfunc, unsigned int reg[4])
| ^
3 errors generated.
error: command 'D:\Programs\Msys2\clang64\bin\gcc.exe' failed with exit code 1
[end of output]

# Bing

The error messages indicate that the variables reg, op, and subfunc are being redefined with different types in your code. This is happening because these variables are declared as unsigned int in the function prototype for cpuid at line 98, but then they are redeclared as int at line 110.

To resolve this issue, you should ensure that the variables are declared with the same types in both places. If the variables should be unsigned int, then the declaration at line 110 should be changed to match. Here's how you might modify the code:

unsigned int reg[4], op, subfunc;

asm volatile("pushq %%rbx \n\t" /\* save %ebx \*/

"cpuid \n\t"

"movl %%ebx, %1 \n\t" /\* save what cpuid just put in %ebx \*/

"popq %%rbx \n\t" /\* restore the old %ebx \*/

: "=a"(reg[0]), "=r"(reg[1]), "=c"(reg[2]), "=d"(reg[3])

: "a"(op), "c"(subfunc)

: "cc");

In this code, reg, op, and subfunc are declared as unsigned int to match the function prototype for cpuid.

Please replace op and subfunc with the actual operation and sub-function numbers in your code. Let me know if you have any other questions! 😊

Source(s)

1. [github.com](https://github.com/intel/kernelflinger/tree/9a84515a0f2b6b5697c49c770513b8c432b6f9d4/libkernelflinger%2Flib.c)

# User

How to use rdrand in python. Give me example.

# Bing

The rdrand instruction is a hardware instruction provided by Intel CPUs for generating random numbers. To use rdrand in Python, you would typically use a Python package that provides a wrapper around the rdrand instruction.

However, please note that using rdrand directly in Python is not common because Python has its own built-in libraries for generating random numbers, such as the random module. The random module uses a pseudo-random number generator that is suitable for most applications.

If you still want to use rdrand, you would need to install a Python package that provides this functionality. As per your previous messages, it seems you have a package named rdrand. Once you have successfully built and installed the rdrand package, you can use it in your Python code like this:

import rdrand

# Generate a random number

random\_number = rdrand.get\_random\_number()

print(random\_number)

In this example, get\_random\_number is a hypothetical function provided by the rdrand package that uses the rdrand instruction to generate a random number. You'll need to replace get\_random\_number with the actual function provided by your rdrand package.

Please note that this is a hypothetical example and the actual usage may vary depending on the specific rdrand package you are using. Always refer to the documentation of the package for accurate usage instructions. Let me know if you have any other questions! 😊

# User

Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: module 'rdrand' has no attribute 'get\_random\_number'

# Bing

The error message indicates that the rdrand module does not have an attribute or function named get\_random\_number. This means that get\_random\_number is not a valid function in the rdrand module.

The available functions depend on how the rdrand module is implemented. You should refer to the documentation or the source code of the rdrand module to see what functions are available.

If you're unsure how to use the rdrand module, you could try using the dir function in Python to list all the attributes and methods of the module. Here's how you can do it:

import rdrand

print(dir(rdrand))

This will print a list of all the attributes and methods in the rdrand module. You can look for a function in this list that generates a random number.

Let me know if you have any other questions! 😊

# User

>>> print(dir(rdrand))

['BaseRandom', 'HAS\_RAND', 'HAS\_SEED', 'RdRandom', 'RdSeedom', 'StdRandom', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'rdrand\_get\_bits', 'rdrand\_get\_bytes', 'rdseed\_get\_bits', 'rdseed\_get\_bytes']

# Bing

From the output of dir(rdrand), it appears that the rdrand module provides the following functions for generating random numbers:

* rdrand\_get\_bits: This function likely returns a random number of a certain bit length.
* rdrand\_get\_bytes: This function likely returns a certain number of random bytes.
* rdseed\_get\_bits: This function likely returns a random seed of a certain bit length.
* rdseed\_get\_bytes: This function likely returns a certain number of random seed bytes.

Here's an example of how you might use the rdrand\_get\_bits function to generate a random number:

import rdrand

# Generate a random number of 32 bits

random\_number = rdrand.rdrand\_get\_bits(32)

print(random\_number)

Please replace 32 with the actual bit length you want. The usage of other functions should be similar. Always refer to the documentation or source code of the rdrand module for accurate usage instructions. Let me know if you have any other questions! 😊

# User

>>> randnum=rdrand.RdRandom()

>>> print(randnum)

<rdrand.RdRandom object at 0x000001CAD0C71F50>

I need to print number is place of above output.

# Bing

The rdrand.RdRandom() call creates an instance of the RdRandom class, which is a random number generator. To get a random number, you need to call a method on this object. If the RdRandom class follows the conventions of Python's built-in random number generators, it should have a method like random(), which returns a random float, or randint(a, b), which returns a random integer between a and b.

Here's how you might use it:

import rdrand

# Create a random number generator

randnum = rdrand.RdRandom()

# Get a random number

random\_number = randnum.random()

print(random\_number)

Or, for a random integer between a and b:

import rdrand

# Create a random number generator

randnum = rdrand.RdRandom()

# Get a random integer between a and b

random\_integer = randnum.randint(a, b)

print(random\_integer)

Please replace a and b with the actual range you want. The usage of other methods should be similar. Always refer to the documentation or source code of the rdrand module for accurate usage instructions. Let me know if you have any other questions! 😊